TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

THESIS NO: 070/MSCS/655

" Multi-Objective Optimization Using Membrane Inspired Evolutionary Algorithm "

by

Bishwo Prasad Lamichhane

A THESIS REPORT

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

April, 2023

Multi-Objective Optimization Using Membrane Inspired Evolutionary Algorithm

by

Bishwo Prasad Lamichhane

070MSCS655

Thesis Supervisor

Sanjeeb Prasad Panday Ph.D

Associate Professor, Department of Electronics and Computer Engineering, Pulchowk Campus

A final thesis report submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer System and Knowledge Engineering

Department of Electronics and Computer Engineering

Institute of Engineering, Pulchowk Campus

Tribhuvan University

Lalitpur, Nepal

April, 2023

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING, PULCHOWK CAMPUS
Department of Electronics and Computer Engineering
M. Sc. in Computer System and Knowledge Engineering Program

Supervisors' Approval Sheet for Student's Thesis Final Defense

Program: M.Sc. in Computer System and Knowledge Engineering (MSCSKE)

Student's Name/Roll No.: Bishwo Prasad Lamichhane *// 070MSCS655*

Title: Multi-Objective Optimization Using Membrane Inspired Evolutionary Algorithm

Supervisor: Dr. Sanjeeb Prasad Pandey

S. No.	Items/Criteria	Review Status	Comments
1	Background Study (Literature Survey & Reviews)	OK	
2	Objective Research Question (Relevancy Justification)	OK	
3	Methodology Design/Formulations		
4	Results I. Testing II. Validation III. Analysis/Discussion	I. OK II. OK III. OK	
5	Resource Utilization (Tool/Data etc.)	Python, PHP, Google Colab	
6	Logical Timeline		Barrier
7	Thesis Integrity (SI < 20%, as attached)		
8	Future Works Outline (if any)		Mentioned in Report

Approval: (YES) / NO

Additional Comments (if any):

Signature

2079/12/26
Date

COPYRIGHT©

DECLARATION

I declare that the work hereby submitted for Master of Science in Computer System and Knowledge Engineering (MSCSKE) at IOE, Pulchowk Campus entitled **"Multi-Objective Optimization Using Membrane Inspired Evolutionary Algorithm"** is my own work and has not been previously submitted by me at any university for any academic award.

I authorize IOE, Pulchowk Campus to lend this thesis to other institution or individuals for the purpose of scholarly research.

Bishwo Prasad Lamichhane
070MSCS655
April, 2023

RECOMMENDATION

The undersigned certify that they have read and recommended to the Department of Electronics and Computer Engineering for acceptance, a thesis entitled **"Multi-Objective Optimization Using Membrane Inspired Evolutionary Algorithm"**, submitted by **Bishwo Prasad Lamichhane** in partial fulfillment of the requirement for the award of the degree of **"Master of Science in Computer System and Knowledge Engineering"**.

.....

Supervisor: Sanjeeb Prasad Pandey, Ph.D.

Associate Professor,

Department of Electronics and Computer Engineering,

Pulchowk Campus, Institute of Engineering, Tribhuvan University.

.....

External Examiner:

.....

Committee Chairperson: Anand Kumar Sah

Program Coordinator, MSc in Computer System and Knowledge Engineering,

Department of Electronics and Computer Engineering,

Institute of Engineering, Tribhuvan University.

Date: April, 2023

DEPARTMENTAL ACCEPTANCE

The thesis entitled **" Multi-Objective Optimization Using Membrane Inspired Evolutionary Algorithm"**, submitted by **Bishwo Prasad Lamichhane** in partial fulfillment of the requirement for the award of the degree of **"Master of Science in Computer System and Knowledge Engineering"** has been accepted as a bonafide record of work independently carried out by his/her in the department.

.....

Dr. Jyoti Tandukar
Head of the Department
Department of Electronics and Computer Engineering,
Pulchowk Campus,
Institute of Engineering,
Tribhuvan University,
Nepal.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to our Head of Department Dr. Jyoti Tandukar, Prof. Dr. Sashidhar Ram Joshi, Prof. Dr. Subarna Shakya, Dr. Aman Shakya, for their encouragement and precious guidance. I am thankful to our program coordinator Anand Kumar Sah for providing suitable platform to prepare this thesis work.

I express my deepest appreciation to Dr. Sanjeeb Prasad Panday, my supervisor, for his guidance, support, and encouragement.

I extend my gratitude to the faculty of Department of Electronics and Computer Engineering and my fellow classmates for sharing their perspectives and offering guidance on my thesis work

Bishwo Prasad Lamichhane
070MSCS655

ABSTRACT

This study adopts membrane computing model to design an evolutionary algorithm to solve multi-objective optimization problems. The algorithm uses various concepts from membrane computing model to create a parallel framework to solve optimization problems. Standard benchmark functions are used to asses the quality of output generated by the algorithm. Output of the algorithm is also compared with standard multi-objective optimization algorithms. The algorithm is capable of generating results comparable to the standard optimization algorithms. This algorithm can also provides multiple parameters to properly utilize its potential to solve different types of optimization problems.

Keywords: Multi-objective optimization, Membrane Computing, Evolutionary Algorithm

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

$2D$	Two Dimensional
DM	Decision Maker
EA	Evolutionary Algorithm
GD	Generational Distance
IDE	Integrated Development Environment
IGD	Inverted Generational Distance
$JSON$	JavaScript Object Notation
LHS	Latin Hypercube Sampling
$MFEA$	Multi-Factorial Evolutionary Algorithm With Online Transfer Parameter Estimation
$MOEA$	Multi-objective Evolutionary Algorithm
$MOEA/D$	Multi-objective Evolutionary Algorithm based on Decomposition
MOO	Multi-objective Optimization
$MOOP$	Multi-objective Optimization Problems
$NSGA$	Non-dominating Sorting Genetic Algorithm
ZDT	Zitzler–Deb–Thiele's Function

CHAPTER 1

INTRODUCTION

1.1 Multi-objective optimization

Multi-objective optimization(MOO) is a field of mathematical optimization problems involving more than one objective functions. It is also called vector optimization, multi-objective programming, multi-criteria optimization or Pareto optimization. Multi-objective optimization problems (MOOP) has been used in various applications where optimal decision is required in the presence of multiple trade-offs. In practical cases where multiple objectives are to be satisfied no single solution exists that is optimal for all the objectives. These type of objectives are referred to as conflicting objectives where improvement of one may degrade the other(s). A solution is called non-dominating or Pareto-optimal if none of the objectives can be improved without degrading the other objectives. Unlike single objective optimization multi-objective optimization usually has more than one solutions that are Pareto-optimal. Purpose of various multi-objective optimization algorithms is to find these non-dominating solution(s).

There are a variety of approaches to solving multi-objective optimization problems, including mathematical programming, evolutionary algorithms, and heuristics. Mathematical programming involves formulating the problem as a set of mathematical equations and solving it using optimization techniques. Evolutionary algorithms, such as genetic algorithms and particle swarm optimization, are inspired by the principles of natural selection and seek to find the best solutions by iteratively improving a population of candidate solutions. Heuristics are problem solving strategies that rely on trial and error rather than a systematic approach. Decision-making methods are also an important part of multi-objective optimization. Once a set of Pareto optimal solutions has been obtained, decision-makers must choose which solution to implement. This is often a complex and

challenging task, as it requires balancing the different objectives against each other and taking into account other factors, such as feasibility and cost.

Multi-objective optimization has a wide range of applications in fields such as engineering, finance, management, and operations research. In engineering, it is used to design products that are both efficient and cost-effective. In finance, it is used to optimize investment portfolios that balance risk and return. In management, it is used to make decisions that balance multiple objectives, such as maximizing profits while minimizing environmental impact.

Non-domination is a key concept in multi-objective optimization that refers to the idea of finding solutions that are not dominated by any other solution in the optimization problem. In multi-objective optimization, the focus is to optimize multiple objectives simultaneously, which can result in a set of optimal solutions, rather than a single solution.

A solution is said to be non-dominated if there does not exist any other solution that is strictly better than it in all objectives. In other words, a non-dominated solution is one that is not dominated by any other solution in the problem. This concept is important because it helps to identify a set of solutions that are considered to be ”optimal” in a multi-objective optimization problem.

The set of all non-dominated solutions in a multi-objective optimization problem is called the Pareto front or Pareto set. Finding this set can be challenging, especially in high-dimensional problems, but it is a crucial step in multi-objective optimization as it helps decision-makers to explore and compare different trade-offs between conflicting objectives.

1.2 Membrane Computing Model

Membrane computing, also known as P system, is a type of computing model that was proposed by Gheorghe Păun in 2000. This computational model is inspired by

biological cells and their complex behavior. Păun's idea was to create a new computing paradigm that could be used to model complex biological systems such as cells and their interactions. The basic idea behind membrane computing is to use a set of membranes or compartments, which are called regions. These regions are used to store and process information in a way that is similar to how cells store and process information. The regions can contain objects such as numbers, symbols, or strings. These objects can interact with each other, move between regions, and be transformed into other objects.

One of the key features of membrane computing is that the regions can have different properties. For example, some regions can be sensitive to certain types of objects or rules, while others can be insensitive to those same objects or rules. This allows for a wide range of computations to be performed, from simple arithmetic to complex pattern recognition and optimization problems. Another important aspect of membrane computing is the use of rules to govern the behavior of the objects and the regions. These rules are called evolution rules and can be applied in different ways depending on the specific problem being solved. For example, the rules can be applied simultaneously to all regions or only to a specific subset of regions.

One of the advantages of membrane computing is its ability to model complex biological systems in a more accurate and efficient way than traditional computational models. This is because membrane computing is able to capture the dynamic behavior of cells and their interactions. Additionally, membrane computing is highly parallel, meaning that multiple computations can be performed simultaneously, leading to faster processing times. Membrane computing has many applications in various fields, including biology, physics, chemistry, and computer science. For example, it can be used to model the behavior of cells and their interactions in biological systems, to optimize supply chain logistics in business, or to optimize resource allocation in computer networks.

Similar to other bio-inspired optimization algorithms membrane computing model can offer greater exploration of solution space and is suitable for Pareto based method of optimization. This method can produce multiple solutions that are close to Pareto-

optimal solutions. With these solutions decision maker (DM) can make suitable decision. Bio-inspired methods are particularly suitable in cases where preferences for objectives (a-priori preference expression) cannot be assigned before optimal solutions are available.

1.3 Problem Statement

Current bio-inspired multi-objective optimization algorithms start with random candidates/population across the entire decision space (some algorithms use preference values from DM), perform some fitness evaluations, and create next generation of population using features of candidates with better fitness values. This process involves processing the entire decision space in sequence which involves processing of all of the population in one go. Partitioning of decision space into hyper volumes can enable parallel execution of evolutionary steps. This results in faster convergence to the global optimum. Computing model proposed by membrane computing can be utilized to achieve that goal.

Generation process of the offspring using current optimization algorithm is done by selection of parent population which are sorted by fitness value, proximity metric and/or other algorithm specific parameters. So two parents of an offspring can be from different regions of the decision space. One region can have differently shaped optimum space than the other region. By partitioning the decision and performing evolutionary steps within the regions can produce better offspring in greater numbers than having the entire decision space as the source of parent population. This reduces the number of generations required to achieve results closer to true Pareto-Front.

1.4 Objectives

Main Objectives

The main objective of this thesis is to create membrane computing inspired evolutionary algorithm to solve multi-objective optimization problem and compare its performance with current bio-inspired algorithms.

Specific Objectives

1. To create evolutionary algorithm capable of solving multi-objective optimization problem Using computational model proposed by membrane computing.
2. Analyze the performance of developed algorithm with following standard MOOP test functions.
   (a) Zitzler–Deb–Thiele's Function 1 (ZDT1)
   (b) Zitzler–Deb–Thiele's Function 2 (ZDT2)
   (c) Zitzler–Deb–Thiele's Function 3 (ZDT3)
3. Compare the performance of developed algorithm with following standard MOOP solving algorithms according to following metric
   (a) Generational Distance (GD)
   (b) Inverted Generational Distance (IGD)

CHAPTER 2

LITERATURE REVIEW

Computing with membranes is a branch of Molecular Computing initiated by Gheorghe Paun by the paper Computing with Membranes[1]. This computing model is also called P systems[2] which have the same computing power with a Turing universal computing model. Membrane systems mainly focuses on the various computational features of the membranes such as transferring chemical substances between membranes or chemical reaction in the region of the membrane, instead of modeling the biological membranes. In strict sense number of principles are abstracted underlying the functioning of biological membranes, and this abstraction is used as the working mechanism of the computing model.

Based on the above-mentioned context, an integral membrane system includes the nested membrane structure, multisets, and reaction rules. Multiset, which consists of a collection of symbol-objects, is placed in the compartments defined by the membrane structure, and it is evolved by executing the reaction rules in a non-deterministic and maximally parallel manner[1]. The membranes except the skin membrane can be dissolved and divided by invoking the corresponding reaction rules. Because the structure of membrane systems provides an enhanced parallelism, membrane systems can solve intractable problems in a polynomial time [2]. More specifically, the membrane system with an enhanced parallelism is able to trade space for time, that is to say, it can solve intractable problems in a feasible time due to making use of an exponential space.

An evolutionary algorithm is proposed in [3] which is based on membrane systems to solve the global numerical optimization problems for single objective functions. The algorithm employs fundamental ingredients of membrane systems, including multisets, reaction rules and membrane structure. In addition, the algorithm incorporates information of the adjacent symbol-objects, to guide the evolution toward the global optimum,

efficiently.

In multi-objective optimization more than one objectives are needed to be reached. When decision maker cannot express the preference information about these objectives such multi-objective optimization methods can be classified as no-preference methods [4].

Legacy works in multi-objective optimization include Weighing Method [4] where weighted sum of all objective functions are merged into one objective and solved as single optimization problem. This will produce one solution among the many Pareto-optimal solutions. Changing of the individual weights can produce other solutions. This method has shortcomings where problem is non-convex. In ε-Constraint Method one of the objective functions is selected to be optimized, the others are converted into constraints [4]. Solving non-convex single-objective optimization problems to global optimality or at least with some guaranteed proximity to the optimal value is known to be a challenge in mathematical optimization [5].

Several evolutionary algorithms are proposed over the years with capabilities to resolve challenges with classical optimization methods [6], [7], [8], [9].

Earlier evolutionary algorithm such as Non-dominating Sorting Genetic Algorithm (NSGA) [6] uses concepts from genetic algorithm such mutation, selection and ranks the results in terms their non-dominance.

NSGA was further improved in NSGA-II [7] and NSGA-III [8] reducing computational complexity and improvement of diversity and optimal values of solutions.

Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [9] combines classical method of optimization known as decomposition with concepts form evolutionary algorithm and is comparable to performance with NSGA-II. The NSGA-II algorithm has also been extended to solve many objective optimization in [10].

Estimation of Pareto Front (PF) and selection of offspring based on the proximity of offspring to the estimated PF is discussed in [11]. Unlike other evolutionary algorithms this method takes the shape of PF into account while exploring solution space. Other

methods to approximate Pareto Front using scalarization methods is outlined in [12].

One of the recent work on optimization based on swarm intelligence is Moth–Flame optimiser [13] where the Moth's swarm is utilized to explore the search space around the Flames set (leader solutions).

Research Gap

Multi-objective optimization is a well-researched field, and many different optimization techniques have been developed to solve multi-objective problems. However, there is still a research gap in the use of membrane computing models for solving multi-objective optimization problems.

Membrane computing is a relatively new computational paradigm that has been shown to be effective in solving a wide range of problems. Membrane computing models are based on the structure and behavior of biological cells and membranes, and they provide a powerful tool for modeling complex systems and solving optimization problems.

Despite the potential benefits of membrane computing models in solving multi-objective optimization problems, there is still a lack of research in this area. Most of the existing research on multi-objective optimization has focused on traditional optimization techniques such as evolutionary algorithms, swarm intelligence, and mathematical programming.

In addition, there is a need for further research on the design and implementation of membrane computing models for multi-objective optimization problems. There are several different types of membrane computing models, each with its own strengths and weaknesses, and further research is needed to determine which models are best suited for different types of optimization problems.

CHAPTER 3

THEORETICAL BACKGROUND

3.1 Multi-objective Optimization Problems

Multi-objective Optimization Problems involves a set of function that are to be maximized or minimized. Mathematically,

$$\text{min } f_m(x), \text{ where } m = 1, 2, 3, ..., M$$

$$\text{subject to}$$

$$g_i(x) \geq 0, \text{ where } i = 1, 2, 3, ..., I$$

$$h_j(x) = 0, \text{ where } i = 1, 2, 3, ..., J$$

$$\text{and } x_L \leq x \leq x_U, \text{ where}$$

- x_L is lower bound and x_U is upper bound
- $f_m(x)$ are objectives to be minimized
- $g_i(x)$ are constraints with inequality
- $h_j(x)$ are constraints

If an objective is to maximize, negation or inverse of the objective becomes minimizing problem. Similar operations can be done to constraints to standardize optimization problem. Multi-objective optimization problems in principle have a set of optimal solutions also known as Pareto-optimal solutions. Goal of any multi-objective optimization algorithm is to find as many and diverse set of Pareto-optimal solutions. In general continuous MOOPs have an infinite number of Pareto optimal solutions (whereas combinatorial MOOPs have a finite but possibly very large number of Pareto optimal solutions) and the Pareto optimal set (consisting of the Pareto optimal solutions) can be non-convex and disconnected.

3.2 Membrane Computing Model

Membrane computing is a field in computer science that focuses on extracting computing concepts and models from the structure and operations of living cells, as well as the organization of cells in tissues and complex structures. The goal is to develop distributed and parallel computing models that can process multiple sets of symbol-objects in a localized manner. This is achieved by encapsulating evolution rules and objects within compartments, which are delimited by membranes. Communication among compartments, including with the external environment, plays a crucial role in this process. Following figure shows the list of bio inspired computing models:

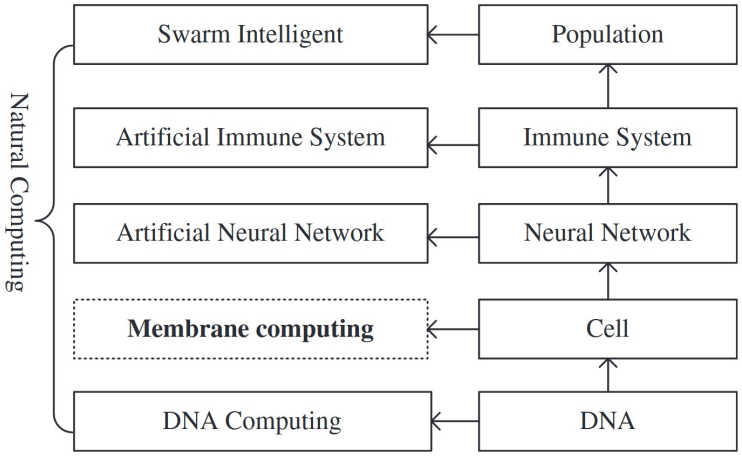


Figure 3.1: Hierarchical model of biological simulation of computing systems

A membrane structure is a hierarchically arranged set of membranes, contained in a distinguished external membrane (usually called the skin membrane). Several membranes can be placed inside the skin membrane (they correspond to the membranes present in a cell, around the nucleus); a membrane without any other membrane inside it is said to be elementary. Each membrane determines a compartment, also called region, the space delimited from above by it and from below by the membranes placed directly inside, if any exists.

Basic structure of cell-like membrane system with degree n

$$\Pi = (V, T, \mu, w_1, w_2, w_3, ..., w_n, R) \tag{3.1}$$

where,

- V is the alphabet. Its element is named as an object. An object is the abstract representation of atomic, molecular or the other chemical substances. The object may be represented by symbol or string, also named as symbol-object
- $T \subseteq V$, where T is the output alphabet
- μ is a membrane structure with degree n
- $w_i \in V^*$, $1 \leq i \leq n$, w_i represents the multi-set in the i^{th} region of the membrane structure μ
- R represents the reaction rules in the region of membranes

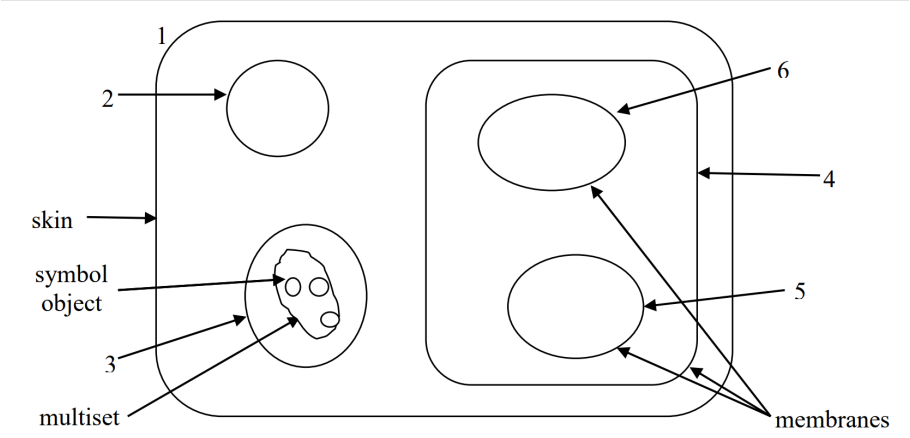


Figure 3.2: Membrane Structure

To solve optimization problems using membrane computing model; symbol objects are used as potential solutions. Then using reaction rule and functional rule as described in eq. 3.1 membrane computing model can be adapted to solve multi-objective optimization problem.

3.3 Initialization Methods

3.3.1 Random Sampling

Random initialization is a commonly used method in optimization problems to generate initial solutions for an optimization algorithm. In this method, a set of potential solutions is randomly generated and used as the starting point for the optimization algorithm. The aim of this method is to explore a wide range of potential solutions and to avoid getting stuck in local optima. Random initialization is particularly useful in optimization problems where the objective function is complex, and the search space is vast. In such cases, generating a random set of initial solutions can increase the chances of finding the global optimum.

One of the key advantages of random initialization is its simplicity. It does not require any prior knowledge or information about the problem, and it is easy to implement. This makes it a popular choice in many optimization algorithms.

However, random initialization also has its limitations. Generating a random set of solutions can be time-consuming, particularly if the search space is large. In addition, there is no guarantee that the initial set of solutions will be a good starting point for the optimization algorithm. In some cases, the initial set of solutions may be too far from the optimal solution, which can lead to a longer optimization process.

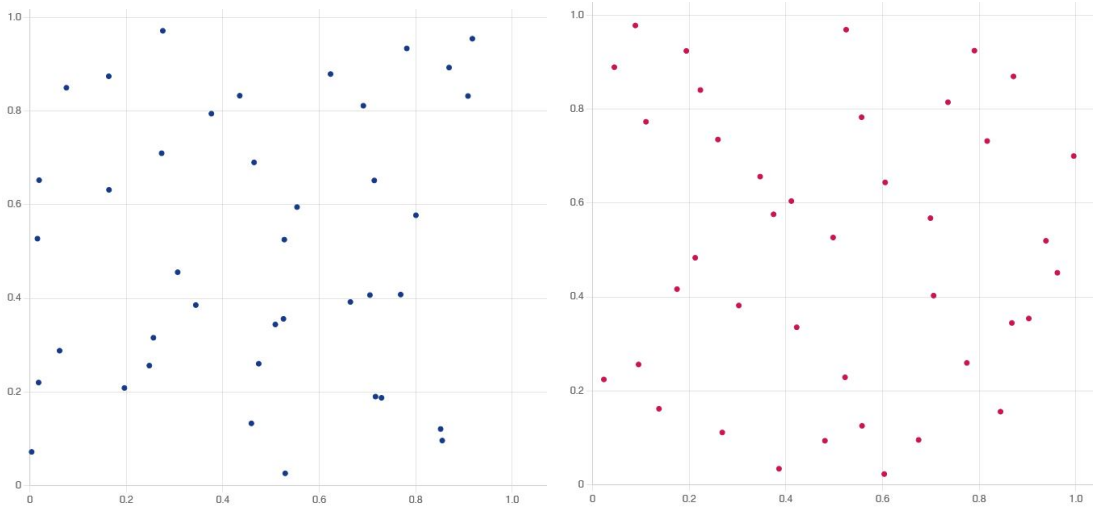
3.3.2 Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) is a another initialization method used in optimization problems that offers an improvement over the traditional random initialization method. LHS is a stratified sampling technique that generates a set of uniformly distributed solutions, while ensuring that each solution is selected from a unique stratum. The technique divides the search space into equally sized strata and ensures that a single solution is selected from each stratum.

LHS offers several advantages over the traditional random initialization method. Firstly, it can produce a more diverse set of solutions, as it ensures that each stratum is represented by at least one solution. This improves the chances of finding the global optimum and reduces the likelihood of getting stuck in a local optimum. Additionally, LHS is computationally efficient and can generate a large number of solutions quickly.

Another key advantage of LHS is its flexibility. The technique can be easily adapted to generate solutions that meet specific requirements. For example, LHS can be used to generate solutions with specific correlations or to ensure that the solutions are evenly distributed across the search space.

Despite its many advantages, LHS also has some limitations. Firstly, it requires knowledge of the search space and the number of solutions to generate. This may not always be available, particularly in complex optimization problems. Additionally, LHS can be challenging to implement for problems with a large number of dimensions or highly non-linear objective functions.



(a) Random Initialization (b) Latin Hypercube Initialization

Figure 3.3: Random Sampling vs Latin Hypercube Sampling

3.4 Benchmark Functions

In MOOP, benchmark functions are commonly used to evaluate and compare the performance of different algorithms. Benchmark functions are mathematical functions that have known optimal solutions and are designed to test the ability of an algorithm to find those solutions.

There are many different benchmark functions available for MOO, and they can vary in terms of their characteristics, such as the number of decision variables, the number of objectives, the complexity of the solution space, and the presence of different types of constraints. Various considerations required to choose effective benchmark functions are

- **Problem dimension**

  The dimensionality of the problem refers to the number of decision variables that need to be optimized. It is important to choose benchmark functions that have a similar dimensionality to the real-world problems that the optimization algorithm will be used to solve.

- **Scalability**

  Benchmark functions should be scalable to different problem sizes, allowing evaluation of the performance of optimization algorithms on problems of varying complexity.

- **Diversity**

  Benchmark functions should be diverse in terms of their structure and characteristics, allowing evaluation the performance of optimization algorithms under different conditions and across a range of problem types.

- **Bias**

  Benchmark functions should be unbiased, so that there are no hidden or artificial constraints that favor one optimization algorithm over another. This will ensure that the performance of optimization algorithms is evaluated fairly and objectively.

With the considerations mentioned above following MOOPs are used as benchmark functions in this study:

- **Zitzler–Deb–Thiele's Function 1**

  ZDT1 is a well-known test function in the field of multi-objective optimization. It was introduced by Zitzler et.al. [14]. The ZDT1 function has two objectives and a variable number of decision variables.

  ZDT1 is a minimization problem with following mathematical structure

  $f_1(x) = x_1$
  $f_2(x) = g(x)h(f_1(x), g(x))$
  $g(x) = 1 + \frac{9}{29}\sum_{i=2}^{30} x_i$
  $h(f_1(x), g(x)) = 1 - \sqrt{\frac{f_1(x)}{g(x)}}$

  Search Domain :

  $0 \leq x_i \leq 1, 1 \leq i \leq 30$

  Pareto front of the ZDT1 function is as follows :

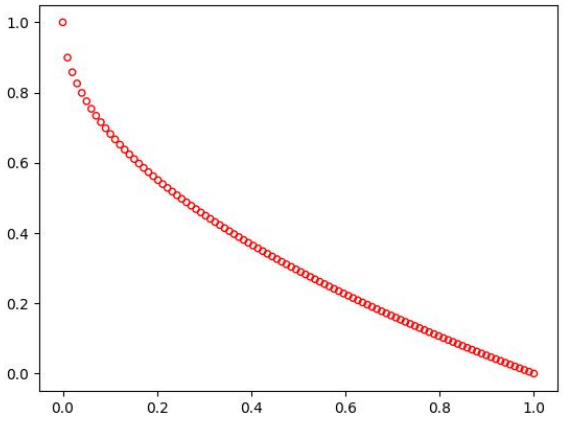


Figure 3.4: True Pareto Front of ZDT1

  The ZDT1 function has a well-defined Pareto front, which is a concave shape. The function is often used as a benchmark problem to test the performance of multi-objective optimization algorithms.

- **Zitzler–Deb–Thiele's Function 2**

  ZDT2 is a minimization problem with following mathematical structure

  $f_1(x) = x_1$
  $f_2(x) = g(x)h(f_1(x), g(x))$
  $g(x) = 1 + \frac{9}{29}\sum_{i=2}^{30} x_i$
  $h(f_1(x), g(x)) = 1 - (\frac{f_1(x)}{g(x)})^2$

  Search Domain :

  $0 \leq x_i \leq 1$
  $1 \leq i \leq 30$

  The ZDT2 function has a non-convex Pareto front, which consists of two disjoint parts. The function is often used as a benchmark problem to test the ability of multi-objective optimization algorithms to handle non-convex Pareto fronts.

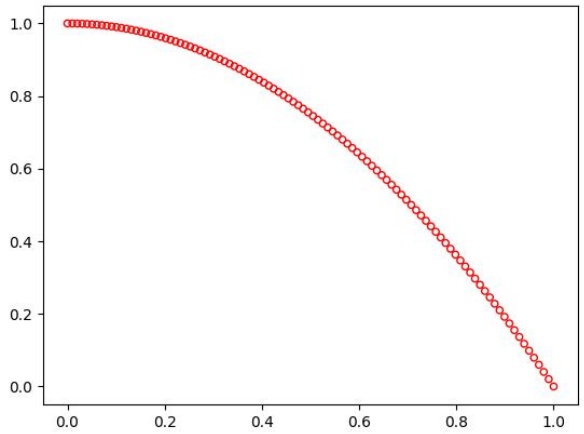


Figure 3.5: True Pareto Front of ZDT2

- **Zitzler–Deb–Thiele's Function 3**

  ZDT3 is a minimization problem with following mathematical structure

  $f_1(x) = x_1$
  $f_2(x) = g(x)h(f_1(x), g(x))$
  $g(x) = 1 + \frac{9}{29}\sum_{i=2}^{30} x_i$
  $h(f_1(x), g(x)) = 1 - \sqrt{\frac{f_1(x)}{g(x)}} - \frac{f_1(x)}{g(x)} sin(10\pi f_1(x))$

  Search Domain :

  $0 \leq x_i \leq 1$
  $1 \leq i \leq 30$

  The ZDT3 function has a convex Pareto front, which makes it a challenging problem for multi-objective optimization algorithms that are designed to handle non-convex Pareto fronts. The function also has a large number of local optima, which makes it difficult to find the global Pareto front.

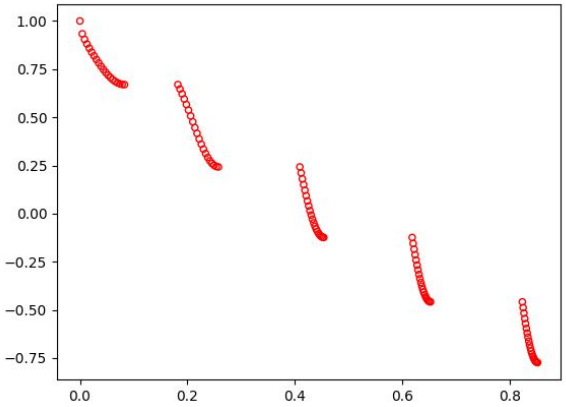


Figure 3.6: True Pareto Front of ZDT3

3.5 Performance Indicators

Performance indicators are used to evaluate and compare the performance of different evolutionary algorithms in solving optimization problems. These indicators are metrics that provide quantitative measures of the quality of the solutions generated by an algorithm, as well as its efficiency. There are many different performance indicators that can be used, depending on the specific problem and the goals of the optimization.

- **Convergence**

  This measures how quickly the algorithm is able to find good solutions to the problem. A faster convergence rate means that the algorithm is able to find better solutions in less time.

- **Coverage**

  This measures how well the algorithm covers the entire search space. A good algorithm should be able to explore all regions of the search space, rather than getting stuck in local optima.

- **Generational Distance**

  Generational Distance(GD) presented in [15] measures the average distance between the solutions generated by the algorithm and the true Pareto front. A smaller generational distance indicates that the algorithm is able to generate solutions that are closer to the true Pareto front.

  GD = $\sqrt{\frac{1}{N}} \sum min(d(x, y))$, for all x in P and y in PF

  where:

  - N is the number of generated solutions
  - P is the set of generated solutions
  - PF is the true Pareto front
  - $d(x, y)$ is the distance between solution x and solution in true Pareto Front y

- $min(d(x, y))$ is the minimum distance between solution x and any solution on the true Pareto front

- **Inverted Generational Distance**

  Inverted Generational Distance(IGD) presented in [16] measures the average distance between the solutions on the true Pareto front and the solutions generated by the algorithm. A smaller inverted generational distance indicates that the algorithm is able to generate solutions that are closer to the true Pareto front.

  IGD = $\sqrt{\frac{1}{N_{pf}}} \sum min(d(x, y))$, for all x in PF and y in P

  where:

  - N_{pf} is the number of solutions in true Pareto front
  - P is the set of generated solutions
  - PF is the true Pareto front
  - $d(x, y)$ is the distance between solution in true Pareto Front x and solution y
  - $min(d(x, y))$ is the minimum distance between solution on the true Pareto front x and any solution y

The main difference between Generational Distance (GD) and Inverted Generational Distance (IGD) is the direction of the comparison between the generated solutions and the true Pareto front.

GD measures the distance from the generated solutions to the true Pareto front, by averaging the distance between each generated solution and its closest solution on the true Pareto front. A lower value of GD indicates better performance, as it means that the generated solutions are closer to the true Pareto front.

On the other hand, IGD measures the distance from the true Pareto front to the generated solutions, by averaging the distance between each solution on the true Pareto front and its closest generated solution. A lower value of IGD indicates better performance, as it means that the generated solutions are better able to cover the true Pareto front.

In summary, while GD measures how well the generated solutions approximate the true Pareto front, IGD measures how well the generated solutions cover the true Pareto front. Both performance indicators are commonly used in multi-objective optimization to evaluate the quality of the generated solutions.

CHAPTER 4

METHODOLOGY

4.1 Evolutionary Optimization Algorithm based on Membrane Computing

This algorithm works by partitioning decision space into multi-dimensional volumes (membranes). Dimension of the membranes depends on the dimension of decision space. Symbol objects (solution, objective pairs) are initialized in each partition. Reproduction and mutation rules are applied inside each membrane. This will double the symbol object count. Then symbol objects are extracted from each membrane and sorted by non-domination. Half of the symbol objects are then rejected and remaining symbol objects are sent into corresponding membranes. At this point some of the membranes may contain more or less symbol objects than at the start. If a membrane contains more symbol objects; division rule is applied to it. If a membrane contains less symbol objects it is merged with its neighbor.

To solve an MOOP with,

$[x_1, x_2, x_3, \ldots, x_n]$ Decision variables,

$[f_1, f_2, f_3, \ldots, f_m]$ Objective functions,

$[l_1, l_2, l_3, \ldots, l_n]$ Lower Bounds and

$[u_1, u_2, u_3, \ldots, u_n]$ Upper Bounds

where,

n is number of decision variable or dimension of the decision space, and

m is number of objective functions or dimension of the solution space

This algorithm takes number of partitions per dimension(p) of decision space, symbol-objects per partition(n_s) and number of iterations/generations(g) as input.

4.1.1 Partitioning of decision space

Partitioning of the decision space is performed to create smaller decision spaces to search for optimum. Number of partitions(p) per dimension is a parameter of the algorithm. Increasing the number of partitions will improve final Pareto front at the cost of increased computation per iteration.

n number decision variables and

p partitions per dimension.

Number of partitions/membranes will be n^p.

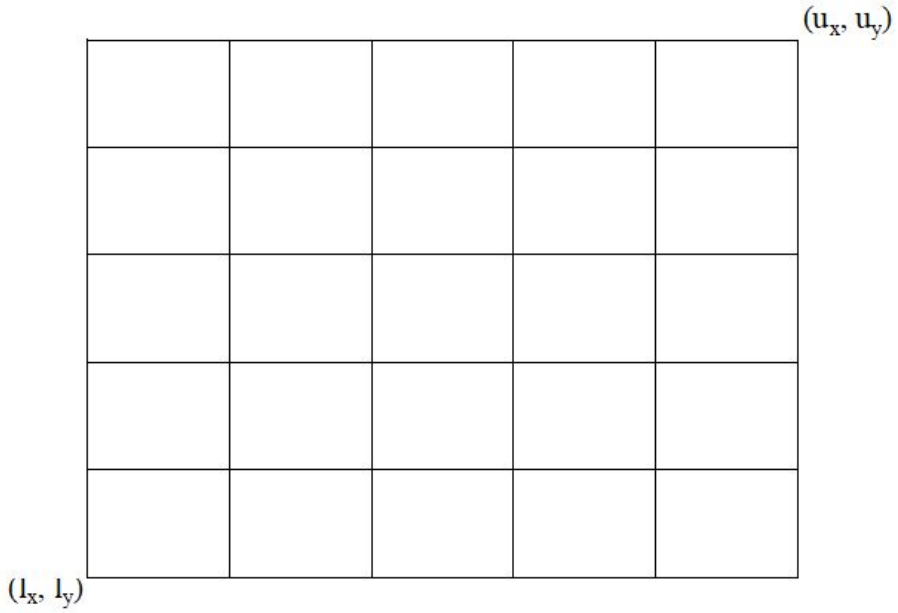


Figure 4.1: Two Dimensional Partitioning of Decision Space

As shown in figure above a two dimensional decision space is partitioned into 5 parts per dimension; in this case number of partitions is $2^5 = 25$.

4.1.2 Initialization

If n_s is the number of symbol objects per membrane then there will be $(n^p) * n_s$ symbol objects. Symbol objects are of the format :

$\{$

'id' : symbolObjectId,

'parentMembrane' : parentMembraneId,

'coordinate' : $[x_1, x_2, x_3, \ldots, x_n]$,

'objectives' : $[f_1, f_2, f_3, \ldots, f_m]$,

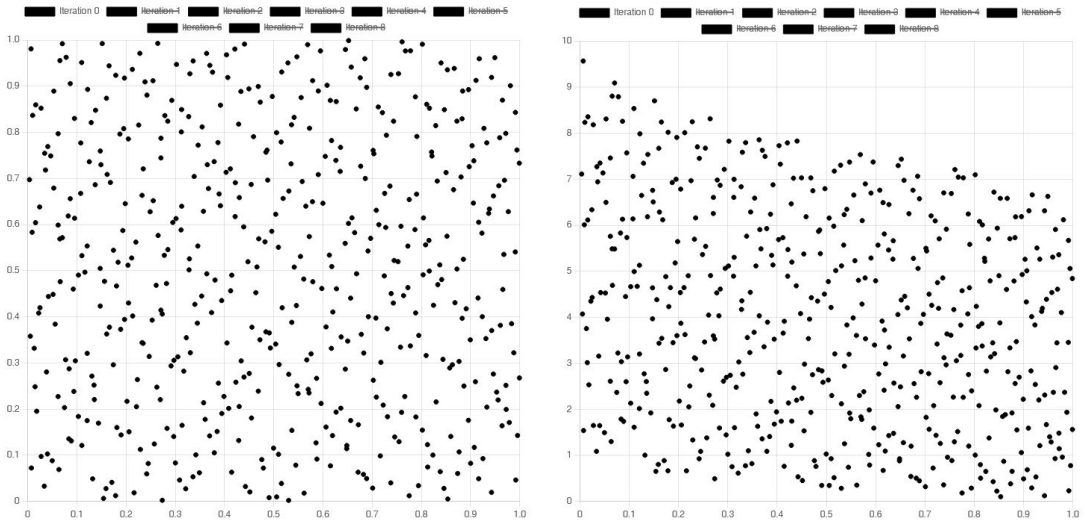
$\}$

where,

n is number of decision variables

m is number of objective functions

To properly explore decision space Latin Hypercube Sampling is used to initialize the first generation of symbol objects.



(a) Initial Symbol Objects

(b) Initial Solutions

Figure 4.2: Initial symbol-objects and solutions

4.1.3 Non-dominating sort inside membrane

A non-dominating sort of symbol objects is performed inside each membranes. Operation inside one membrane is independent of another membrane so this step can be performed for all of the membranes simultaneously. Result of this step is a set of fronts for each membrane. Output of this step is of the form

{
'front-0' : [$symbolObjectId_1$, $symbolObjectId_2$, ..., $symbolObjectId_m$]
'front-1' : [$symbolObjectId_1$, $symbolObjectId_2$, ..., $symbolObjectId_n$]
.....
'front-p' : [$symbolObjectId_1$, $symbolObjectId_2$, ..., $symbolObjectId_o$]
}

Front 0 – No other solution dominate the solution in this front
Front 1 – Solutions in this front are dominated in exactly one objective by solution in front 0 and so on.

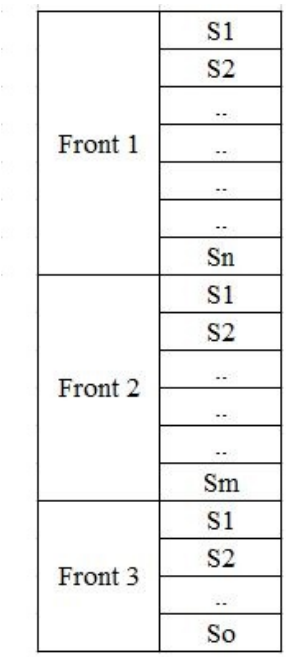


Figure 4.3: Result of Non-dominating Sort

4.1.4 Mate Assignment

Sorted symbol objects are divided into two parts elite solutions and non-elite solutions. A parent pair is generated by selecting one from the elite solutions and one from all of the solutions.

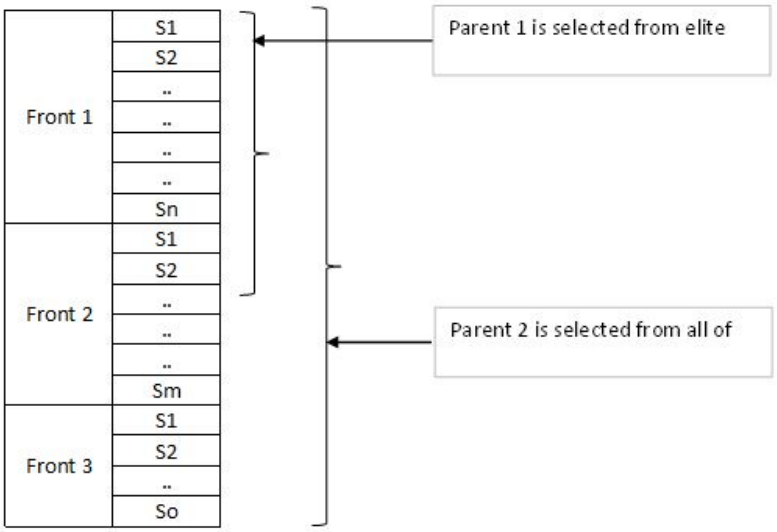


Figure 4.4: Parent Selection

Elite parent is selected from top half of the symbol objects. Second parent is selected from entire population inside the membrane.

This step is independent of another membrane so this step can be performed for all of the membranes simultaneously. Result of this step is a set of symbol objects pair for each membrane. Output of this step is of the form:

{

'pair-1' : ['parent-1', 'parent-2']

'pair-2' : ['parent-1', 'parent-2']

.....

'pair-n_s' : ['parent-1', 'parent-2']

}

4.1.5 Reproduction

An offspring is created by combining the features of parent pair. If one of the parent is dominating the other; offspring gets more characteristics from dominating parent. Otherwise offspring gets equal characteristics from both parents. Small mutation is introduced into offspring to preserve diversity. To generate i^{th} dimension of decision variable of offspring :

Given,

Mutation Chance = m

Elite Parent weight = w_e, this is one of the algorithm parameter and must be ≥ 0.5

Parent 1 Weight : w_1

Parent 2 Weight : w_2

Then, weight for parent 1 for i^{th} dimension of decision variable is

$$w_{1i} = \begin{cases} 0.5 + \text{Rand}(-m, +m), & \text{if Parent 1 and Parent 2 are in same front} \\ w_e + \text{Rand}(-m, +m), & \text{if Parent 1 dominates Parent 2} \end{cases} \tag{4.1}$$

$$w_{2i} = 1 - w_{1i} \tag{4.2}$$

$$x_i = w_{1i} * x_{1i} + w_{2i} * x_{2i} \tag{4.3}$$

In matrix form :

$$\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} w_{11} & 0 & 0 \\ 0 & w_{12} & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & w_{1n} \end{bmatrix} \times \begin{bmatrix} x_{11} \\ x_{12} \\ \cdots \\ x_{1n} \end{bmatrix} + \begin{bmatrix} w_{21} & 0 & 0 \\ 0 & w_{22} & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & w_{2n} \end{bmatrix} \times \begin{bmatrix} x_{21} \\ x_{22} \\ \cdots \\ x_{2n} \end{bmatrix} \tag{4.4}$$

where,

x_{1i} is decision variable value of i^{th} dimension of parent 1

x_{2i} is decision variable value of i^{th} dimension of parent 2

4.1.6 Global non-dominating sort

The symbol objects are extracted from all of the membranes. Extracted symbol objects are then sorted according to non-domination. This operation generates a number of Pareto-fronts.
Front 0 – No other solution dominate the solution in this front
Front 1 – Solutions in this front are dominated in exactly one objective by solution in front 0 and so on.
At this stage termination condition is checked. Termination condition includes number of iteration and first front solution count. If termination condition is met then front-0 is the result of optimization. If not algorithm proceeds to steps below.

4.1.7 Reject Rule

In this step half of the low ranking solutions (solution in higher fronts) are rejected. If a front is being partially rejected, then symbol objects in this front are rejected in such a manner such that every membrane gets similar number of symbol objects as much as possible. This will ensure diversity of solutions.

4.1.8 Repopulate Rule

Accepted symbol objects are sent into their corresponding parent membranes.

4.1.9 Divide and Merge Rule

At this point some of the membranes may contain more or less symbol objects than at the start. If a membrane contains more symbol objects; division rule is applied to it. If a membrane contains less symbol objects it is merged with its neighbor.

Threshold for division $thresholdDivide$ = 150%
Threshold for merge $thresholdMerge$ = 50%

At the end of this step number of membranes and number of symbol objects are same

as they are at the initialization step. But number of symbol objects inside are between $thresholdMerge$ to $thresholdDivide$

When a membrane is divided into two membranes; symbol objects from original membrane is assigned to new membranes by dividing equally both elite and non-elite solutions.

Repeat from ”Non-dominating sort inside membrane” step until termination condition is reached.

4.2 Flowchart

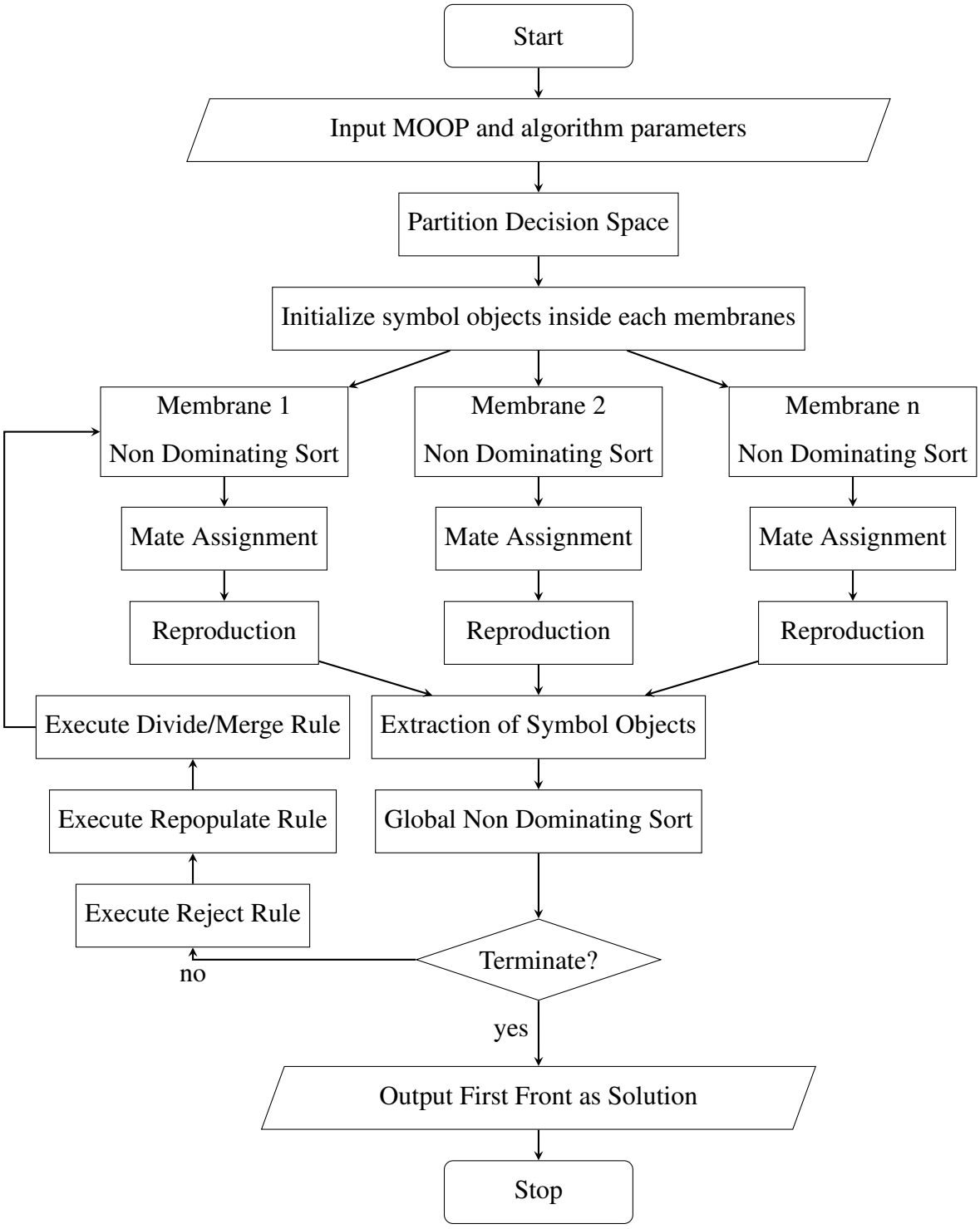


Figure 4.5: Flowchart : Evolutionary Optimization Algorithm based on Membrane Computing

4.3 Complexity of the algorithm

Most expensive operation of this algorithm is Global non-dominating sort where all of the symbol objects from each membranes are extracted and sorted. The computational complexity of Fast Non-Dominating Sort (FNS) is $O(MN^2)$, where M is the number of objectives and N is the number of solutions to be sorted. In the worst case scenario where all solutions are non-dominated, the complexity will be $O(MN^2)$.

In terms of the parameters of this algorithm

Complexity = $O(M(n^p * n_s)^2)$

where

n is number of decision variable or dimension of the decision space, and

M is number of objective functions or dimension of the solution space

p partitions per dimension.

n_s is the number of symbol objects per membrane

4.4 Tools and IDE

Table 4.1: Programming language and libraries used in this study.

SN	Programming Language/IDE	Version	Usage
1	Python	3.11.1	To use benchmark function and indicators
2	pymoo	0.6.0.1	MOOP open source implementations
3	Numpy	1.24.2	For mathematical calculation
4	PHP	8.2.0	For the implementation of algorithm presented in this thesis
5	Netbeans IDE	17	As a coding platform for PHP language
6	PyCharm IDE	2022.3.3	As a coding platform for python language

CHAPTER 5

RESULTS AND ANALYSIS

This section outlines the working of the algorithm, experimental setup, analysis of parameter sensitivity, analysis of the optimization result and comparison analysis.

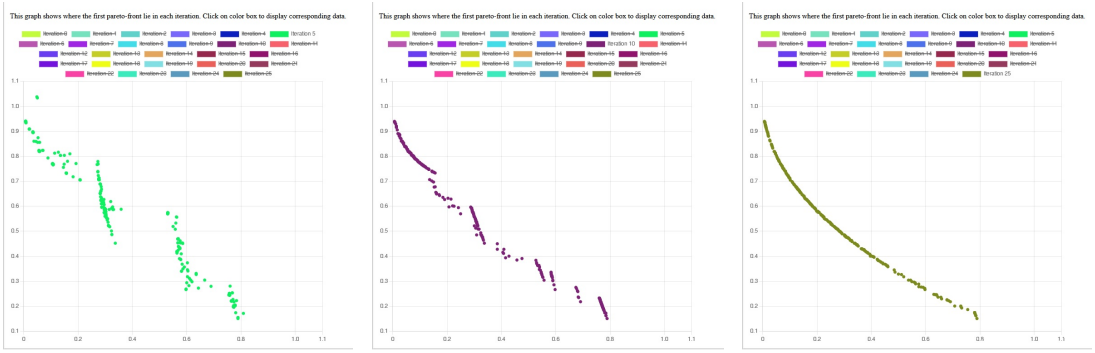
5.1 Experimental setup

All of the programming, testing, debugging, execution and bench-marking of MOOP solving algorithms are done in computer setup with:

- Intel Core I5 5200U processor (2 physical cores)
- 8 GB of DDR3 RAM

5.2 Algorithm Output

The algorithm performs effectively on the benchmark functions presented in section 3.4. The utilization of partitions facilitates rapid convergence of generated solutions to the true Pareto front, resulting in a well-distributed output in the objective space. The accompanying figure illustrates the evolution of solutions towards the true Pareto front.



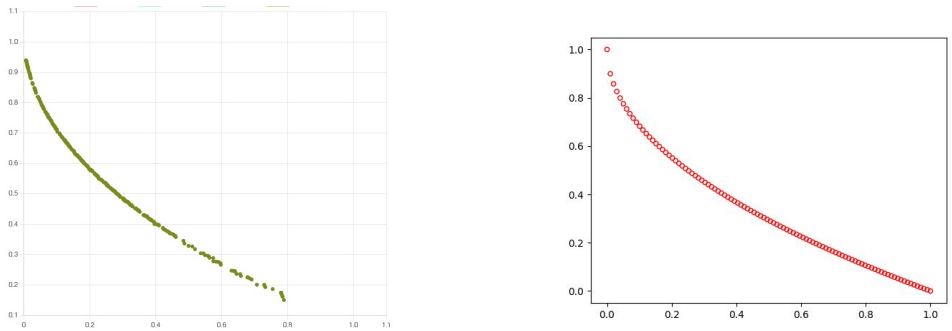
(a) Iteration 5 (b) Iteration 10 (c) Iteration 25

Figure 5.1: Progression of Pareto-Front

The depicted figure demonstrates the clustering of the Pareto front in the early iterations, which can be attributed to the partitioning of the decision space and the application of evolutionary rules within those partitions. However, the global non-domination sort and reject rule work to smooth out these clusters.

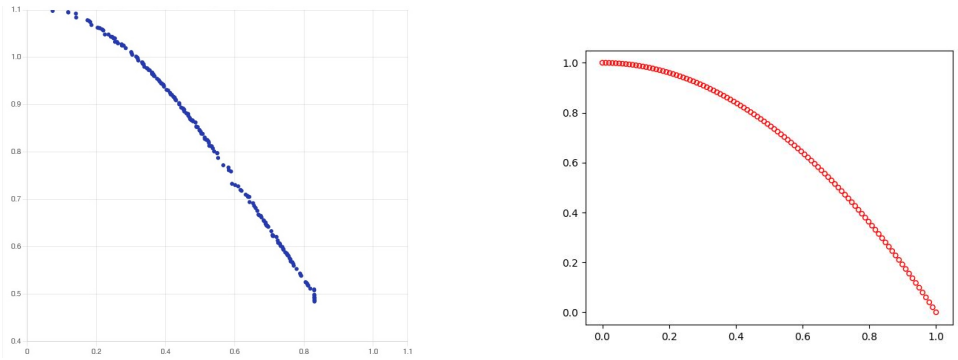
By partitioning the decision space this algorithm is able to reduce crowding of the solution.

The algorithm is generating satisfactory results with benchmark objectives.



(a) Output of the Algorithm (b) True Pareto front of ZDT1

Figure 5.2: Output of the algorithm for ZDT1 test objective



(a) Output of the Algorithm (b) True Pareto front of ZDT2

Figure 5.3: Output of the algorithm for ZDT2 test objective

(a) Output of the Algorithm

(b) True Pareto front of ZDT3

Figure 5.4: Output of the algorithm for ZDT3 test objective

Figure 5.2, 5.3 and 5.4 shows the output of the membrane computing inspired evolutionary algorithm. As seen from the figures the algorithm is capable of producing satisfactory output that are close to the true pareto front.

In figure 5.2 pareto front of ZDT1 is shown. This test objective has continuous pareto front with concave shape. This is one of the simplest test objective to optimize.

In figure 5.3 pareto front of ZDT2 is shown. This test objective has continuous pareto front with convex shape. Due to the convex shape; this objective is slightly difficult to achieve.

In figure 5.4 pareto front of ZDT3 is shown. This test objective has dis-continuous pareto front.

These figures are to illustrate the output of the algorithm in terms of how closely the pareto fronts of the output resemble the true pareto front.

5.3 Performance Analysis

As an evolutionary algorithm, this method is non-deterministic in nature. Therefore, to accurately evaluate the quality of the solutions generated, statistical analysis was conducted across multiple executions.

5.3.1 Algorithm output with varying problem complexity

Setup :

- Problem : ZDT1
- Number of partitions : 4
- Symbol objects per partition : 32

Following table illustrates the effect on quality of output by increasing dimension of decision space

Table 5.1: Performance of Algorithm with changing dimension

SN	Dimension	Performance Indicators							
		GD				IGD			
		Best	Worst	Mean	Std. Dev.	Best	Worst	Mean	Std. Dev.
1	2	3.54E-03	1.51E-02	7.18E-03	2.67E-03	3.48E-03	1.15E-02	6.36E-03	1.91E-03
2	3	1.60E-02	5.82E-02	3.51E-02	1.10E-02	1.49E-02	5.77E-02	3.45E-02	1.11E-02
3	4	4.37E-02	8.48E-02	6.89E-02	1.22E-02	4.18E-02	8.46E-02	6.83E-02	1.24E-02

As dimension of the problem increases; output of the algorithm gets further away from true Pareto front.

Following table illustrates the effect on quality of output by increasing number of partitions per dimension

Table 5.2: Performance of Algorithm with changing partitions

SN	Partitions	Performance Indicators							
		GD				IGD			
		Best	Worst	Mean	Std. Dev.	Best	Worst	Mean	Std. Dev.
1	6	2.39E-03	1.03E-02	5.82E-03	2.31E-03	2.11E-03	9.72E-03	5.16E-03	1.99E-03
2	5	2.72E-03	1.02E-02	5.93E-03	2.21E-03	2.28E-03	9.40E-03	5.34E-03	1.87E-03
3	4	4.69E-03	1.02E-02	5.93E-03	2.21E-03	2.28E-03	9.40E-03	5.34E-03	1.87E-03
4	3	4.71E-03	1.70E-02	9.71E-03	3.14E-03	4.55E-03	1.35E-02	8.37E-03	2.51E-03
5	2	5.85E-03	1.39E-02	1.07E-02	2.51E-03	5.71E-03	1.37E-02	1.01E-02	2.45E-03

As partitions per dimension is increased; output of the algorithm gets further closer to true Pareto front.

5.4 Comparison with standard MOOP solvers

To compare the output of the algorithm with standard MOOP solvers, following setup was created :

Optimization Problem

Objective : ZDT1

Decision space Dimension = 2

Lower Bound = 0

Upper Bound = 1

Parameters for membrane inspired algorithm

Number of partitions per dimension = 4

Number of symbol objects per partitions = 32

Parameters for NSGA-II

Number of population = 512

Parameters for MOEA/D

Subproblems: N = 512,

Number of neighbors: T = 10

The probability of selecting parents from the neighbors = 0.9

Mutation rate: C R = F = 0.5

The performance indicators of each algorithm were calculated based on 20 independent runs using the specified parameters. The statistical analysis of the results is presented below:

Table 5.3: Statistical Results for GD

GD	Mem. Algorithm	NSGA II	MOEA/D
ZDT1			
Best	**3.05E-03**	4.69E-03	4.44E-03
Worst	1.02E-02	7.54E-03	**7.13E-03**
Mean	6.32E-03	6.31E-03	**5.97E-03**
Std. Dev.	1.86E-03	8.84E-04	**8.38E-04**
ZDT2			
Best	6.18E-03	5.19E-03	**4.38E-03**
Worst	2.39E-02	**8.10E-03**	8.16E-03
Mean	1.50E-02	6.85E-03	**6.32E-03**
Std. Dev.	4.89E-03	**7.43E-04**	8.37E-04
ZDT3			
Best	**2.08E-03**	5.21E-03	4.72E-03
Worst	1.76E-02	**7.57E-03**	7.46E-03
Mean	9.79E-03	6.36E-03	**5.95E-03**
Std. Dev.	4.45E-03	7.54E-04	**8.68E-04**

Bold values indicate best results

The algorithm presented in this thesis was able to generate results that are closer to the standard MOOP algorithms. Some cases proposed algorithm has performed better than the standard MOOP solvers. But mean and standard deviation values of GD generated by the proposed algorithm is still worse than the other algorithms indicating larger variance in the output generated.

Table 5.4: Statistical Results for IGD

IGD	Mem. Algorithm	NSGA II	MOEA/D
ZDT1			
Best	2.65E-03	**1.27E-03**	2.65E-04
Worst	8.76E-03	**3.79E-03**	4.03E-03
Mean	5.94E-03	2.71E-03	**2.19E-03**
Std. Dev.	1.66E-03	**7.94E-04**	8.99E-04
ZDT2			
Best	6.14E-03	3.71E-03	**3.02E-03**
Worst	2.37E-02	6.66E-03	**6.31E-03**
Mean	1.46E-02	5.09E-03	**4.64E-03**
Std. Dev.	4.68E-03	**9.32E-04**	9.77E-04
ZDT3			
Best	**1.39E-03**	6.15E-04	2.65E-04
Worst	1.74E-02	3.43E-03	**3.07E-03**
Mean	9.60E-03	1.97E-03	**1.55E-03**

Bold values indicate best results

The algorithm presented in this thesis was able to generate results that are closer to the standard MOOP algorithms. Some cases proposed algorithm has performed better than the standard MOOP solvers. But mean and standard deviation values of IGD generated by the proposed algorithm is still worse than the other algorithms indicating larger variance in the output generated.

CHAPTER 6

Conclusion and Future Work

6.1 Summary

Algorithm presented in this thesis is capable of solving MOOP of different complexity. Quality of the solutions generated by this algorithm is comparable to current state-of-the art algorithms. This algorithm also presents a way to use the parallel computational resource by partitioning and parallel execution of evolutionary rules.

6.2 Conclusion

This study involved the development of an evolutionary algorithm based on the membrane computing model, which facilitated the parallel execution of evolutionary stages on a large scale. The algorithm demonstrated the ability to solve multi-objective optimization problems of varying complexity, and was applied to solve benchmark optimization problems. To evaluate the algorithm's performance, well-balanced performance indicators were utilized, and its output was compared to that of standard MOOP solvers. Additionally, statistical analysis of the algorithm's output and comparison results were presented.

6.3 Future Work

Several concepts from membrane computing can be incorporated to better the quality of solutions. To adapt the algorithm for higher dimension of decision space; utilization of additive nature of MOOP as presented in [17] can be done. Candidates who failed in mating selection stage can be agitated using brownian motion method as suggested in [18] to better utilize the population. Finally this algorithm can be adapted to solve multi-objective integer and mixed-integer programming problems with hybrid approach proposed in [19].

REFERENCES

[1] G. Paun, "Computing with membrane," *J. Comput. System Sci.*, vol. 61, pp. 108–143, 2000.

[2] G. Paun and G. Rozenberg, "A guide to membrane computing," *Theory Computer Sci.*, vol. 287, pp. 73–100, 2002.

[3] M. Han, C. Lui, and J. Xing, "An evolutionary membrane algorithm for global numerical optimization problems," *Elsevier Information Sciences*, vol. 276, 2014.

[4] J. Branke, K. Deb, K. Miettinen, and R. S. (Eds.), "Multiobjective optimization: Interactive and evolutionary approaches," *Applied Soft Computing*, 2008.

[5] G. Eichfelder, "Twenty years of continuous multiobjective optimization in the twenty-first century," *2019 2nd International Conference on new Trends in Computing Sciences (ICTCS)*, pp. 1–6, 2019.

[6] N. Srinivas and K. Deb, "Multiobjective function optimization using nondominated sorting genetic algorithms," *Evolutionary Computing*, vol. 2, no. 3, pp. 221—248, 1995.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, p. 187, 2002.

[8] K. Deb, H. Jain, and R. S. nski (Eds.), "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, 2014.

[9] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on

decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, 2007.

[10] M. Elarbi, S. Bechikh, A. Gupta, L. B. Said, and Y.-S. Ong, "A new decomposition-based NSGA-II for many-objective optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 7, pp. 1191–1210, Jul. 2018. [Online]. Available: https://doi.org/10.1109/tsmc.2017.2654301

[11] A. Panichella, "An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization," *In Genetic and Evolutionary Computation Conference*, 2019.

[12] R. S. Burachik, C. Y. Kaya, and M. M. Rizvi, "A new scalarization technique and new algorithms to generate pareto fronts," *SIAM Journal on Optimization*, vol. 27, no. 2, pp. 1010–1034, Jan. 2017. [Online]. Available: https://doi.org/10.1137/16m1083967

[13] D. Zouache, F. Abdelaziz, M. Lefkir, and N. Chalabi, "Guided moth–flame optimiser for multi-objective optimization problems," *Annals of Operations Research*, vol. 296, 01 2021.

[14] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: empirical results," *Evolutionary computation*, vol. 8, no. 2, pp. 173–195, 2000.

[15] D. A. V. Veldhuizen, "Multiobjective evolutionary algorithms: classifications, analyses, and new innovations." *Technical Report, Evolutionary Computation*, 1999.

[16] H. Ishibuchi, H. Masuda, Y. Tanigaki, and Y. Nojima, "Modified distance calculation in generational distance and inverted generational distance," in *Lecture Notes in Computer Science*. Springer International Publishing, 2015, pp. 110–125. [Online]. Available: https://doi.org/10.1007/978-3-319-15892-1_8

[17] H. Wang, H. Xu, and Y. Yuan, "High-dimensional expensive multi-objective optimization via additive structure," *Intelligent Systems with Applications*, vol. 14, p. 200062, May 2022. [Online]. Available: https://doi.org/10.1016/j.iswa.2022.200062

[18] K. Sowjanya and S. K. Injeti, "Investigation of butterfly optimization and gases brownian motion optimization algorithms for optimal multilevel image thresholding," *Expert Systems with Applications*, vol. 182, p. 115286, Nov. 2021. [Online]. Available: https://doi.org/10.1016/j.eswa.2021.115286

[19] R. S. Burachik, C. Y. Kaya, and M. M. Rizvi, "Algorithms for generating pareto fronts of multi-objective integer and mixed-integer programming problems," *Engineering Optimization*, vol. 54, no. 8, pp. 1413–1425, Jun. 2021. [Online]. Available: https://doi.org/10.1080/0305215x.2021.1939695