CLOUD SERVICE SELECTION AND WORKFLOW SCHEDULING USING P SYSTEM

Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

SANTHANAM RAGHAVAN

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575025

JUNE, 2021

DECLARATION

By the Ph.D. Research Scholar

I hereby *declare* that the Research Thesis entitled **CLOUD SERVICE SELECTION AND WORKFLOW SCHEDULING USING P SYSTEM** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfillment of the requirements for the award of the Degree of **Doctor of Philosophy** in **Computer Science and Engineering** is a *bonafide report of the research work carried out by me*. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

SANTHANAM RAGHAVAN

Reg. No. 155131 CS15FV12

Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: June 1, 2021

CERTIFICATE

This is to *certify* that the Research Thesis entitled **CLOUD SERVICE SELECTION AND WORKFLOW SCHEDULING USING P SYSTEM** submitted by **SANTHANAM RAGHAVAN**, (Reg. No. 155131 CS15FV12) as the record of the research work carried out by him, is *accepted as the Research Thesis submission* in partial fulfillment of the requirements for the award of degree of **Doctor of Philosophy**.

(Prof. K. Chandrasekaran) 01/06/2021

Research Supervisor

Dr. K. CHANDRA SEKARAN Ph.D.
Professor, Dept. of Computer Sc. & Engg.
National Institute of Technology Karnataka
Surathkal, Srinivasnagar - 575 025
Mangalore, Karnataka State, India.

17/06/2021

Chairman - DRPC

(Signature with Date and Seal)

ACKNOWLEDGEMENT

First and foremost, I would thank and express my sincere gratitude to my Research Supervisor, Prof. K. Chandrasekaran for believing in me and accepting me as his student. I will always be indebted to his guidance, immeasurable help and support that he has offered me throughout my stay at NITK. I have tremendously gained from his profound knowledge and have learned a lot from him technically and otherwise, which I feel is to be practised throughout my life. His remarkable perception of subjects and his hard work always motivates me to work and learn. Without his support and blessings I wouldn't have made it this far and completed my thesis. I consider myself to be truly blessed to be associated with him.

My sincere thanks to Dr. Mohit P. Tahiliani, Department of CSE, RPAC Member, for his constructive comments that have helped me in improvising my work. I thank him for his constant support technically and otherwise throughout my stay at NITK. I would like to thank, Prof. N. S. V. Shet, Department of ECE, RPAC member, for comments that have allowed me to enhance this work.

I extend my sincere thanks to the Head of the Department for his continuous support from the department. My sincere thanks to Prof. Santhi Thilagam, Department of CSE, for her support throughout my stay at NITK. I sincerely thank all the faculties of Department of CSE, who have graciously helped me technically and otherwise. I extend my sincere thanks to all the supporting staff who have provided all the support required for the work. I specially thank all my friends and colleagues at NITK whose support and company has made my stay memorable. A sincere thanks to all those who have supported me directly or indirectly in this work.

Finally, a big thanks to my family for their patience and unconditional support.

Place: NITK, Surathkal

Date: June, 2021 Santhanam Raghavan

ABSTRACT

Cloud Computing is a decade old technology that has changed the landscape of the internet based business model. This technology manifested itself unheralded, a decade ago and has been growing since. It now stands with several inherent complex problems, as a result of its expansion. Out of several issues being researched, service selection in cloud is one of the prime issues which is getting primary attention. Service selection is a process of selecting (ranking) services from a pool of available cloud services which is often based on multiple Quality of Service (QoS) attributes. Our work is divided into two major components. The first part of our work is solving the problem of cloud service selection. This study proposes inherently parallel, robust models for service selection in cloud based on a natural computing model called membrane computing. Membrane Computing, which is realised using P Systems, is an inherently parallel model that is based on the concept of animal cell interaction. There are several variants of P Systems and here Enzymatic Numerical P System (ENPS) is used, based on its suitability to the problem being solved. Multiple approaches have been proposed and the results are analysed. Additionally, two new software tools required for ENPS execution are proposed. The second part involves designing and implementing the algorithm for workflow scheduling in cloud. Workflow is a group of tasks that are collectively aimed at doing a single work. Cloud workflows consist of tasks to be mapped to Virtual Machines (VMs) that are part of the cloud. The process of assigning limited number of VMs to the tasks in a particular manner to optimize certain quality factor, is referred to as workflow scheduling in cloud. In this study the effort is to minimise makespan, which is the net time taken by the workflow to get executed. The ENPS model is used to obtain the sequence of the schedule, based on which the makespan is calculated and compared with other standard methods.

Keywords: Cloud Computing, Cloud Service Selection, P System, Enzymatic Numerical P System, Workflow Scheduling.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

ACO	Ant Colony Optimization
AHP	Analytic Hierarchy Process
ANP	Analytic Network Processing
BaaRS	Balanced and file Reuse-Replication Scheduling
BRS	Best Resource Selection
BTS	Balanced Time Scheduling
CAES	Cost and Energy Aware Scheduler
CLIPS	C Language Integrated Production System
CSMIC	Cloud Service Measurement Index Consortium
CSV	Comma Separated Values
CUDA	Compute Unified Device Architecture
DSL	Domain Specific Language
EFT- MER	Earliest Finish Time - Maximum Effective Reduction
EFT-MER-EL	Earliest Finish Time - Maximum Effective Reduction Exploring Laxity Time
EIPR	Enhance IC-PCP with Replication
ENPS	Enzymatic Numerical P System

ENPS-IAHP	Enzymatic Numerical P System based Improved Analytic Hierarchy Process
ENPS-IPROMETHEE	Enzymatic Numerical P System based Improved Preference Ranking Organization Method for Enrichment Evaluation
ENPS-ITOPSIS	Enzymatic Numerical P System based Improved Technique of Order Preference using Ideal Solution
ENPS-MTOPSIS	Enzymatic Numerical P System based Modified Technique of Order Preference using Ideal Solution
EWSA	Efficient Workflow Scheduling Algorithm
FAHP	Fuzzy Analytic Hierarchy Process
FCFS	First Come First Serve
FMPSO	Fuzzy System and Modified PSO
FSV	Flexible Selection of VMs
FWS	Fault Tolerant Workflow Scheduling
GPU	Graphic Processing Unit
GUI	Graphical User Interface
HEFT	Heterogeneous Earliest Finish Time
HPSO	Hybrid Particle Swarm Optimization
HTML	Hypertext Markup Language
IaaS	Infrastructure as a Service
IAHP	Improved Analytic Hierarchy Process
IC-PCP	IaaS Cloud Partial Critical Paths

IOO	Iterative Ordinal Optimization
IWD	Intelligent Water Drop
IWD-DWS	Intelligent Water Drop - Dynamic Workflow Scheduling
KTDR	Kendall Tau Distance Ratio
L-ACO	L-Ant Colony Optimization
MAUT	Multi-Attribute Utility Theory
MCDM	Multi-criteria Decision Making
MCT	Minimum Completion Time
MeCoSim	Membrane Computing Simulator
MoACS	Multi-Objective Ant Colony System
MOHEFT	Modified Heterogeneous Earliest Finish Time
MPI	Message Passing Interface
MSLBL	Minimising Schedule Length using Budget Level
MVC	Model View Controller
NAHP	Neutrosophic Analytic Hierarchy Process
NIST	National Institute of Standards and Technology
NPS	Numerical P System
NSGA	Non-Dominated Sorted Genetic Algorithm
PaaS	Platform as a Service
PBTS	Partition Balancing Time Scheduling
PCA	Peer Cloud Assisted

PMCGPU	Parallel Simulators for Membrane Computing on the GPU
PROMETHEE	Preference Ranking Method for Enrichment Evaluation
PSO	Particle Swarm Optimization
QoS	Quality of Service
QuaRAM	QoS Aware Cloud Application Management
RDPSO	Revised Discrete Particle Swarm Optimization
RMI	Remote Method Invocation
RRT	Rapidly Exploring Random Tree
SaaS	Software as a Service
SBML	Systems Biology Markup Language
SimCM	Simulador de Computacion con Membranes
SMI	Service Measurement Index
SNUPS	Simulator for Numerical P System
TCR	Task Completion Rate
TOPSIS	Technique of Order Preference using Ideal Solution
UDP	User Datagram Protocol
VM	Virtual Machine
VRTR	VMs Reserve Time Rate
XML	eXtensible Markup Language

Chapter 1

INTRODUCTION

Cloud Computing is one of the most used and also one of the most popular technologies that is available today. There are several components of cloud computing that are being continuously enhanced and researched upon. Among these, two important topics are Service Selection and Workflow Scheduling in Cloud. From time to time, there have been several advancements in these areas. The thesis work deals with such advancement in both the areas. The first work is around service selection in cloud where membrane-based algorithms for service selection are proposed. The proposed methods are implemented and results are analysed. Further two tools assisting the implementation are proposed. These two tools are also stand alone contributions to the membrane computing community. The next work, namely, Workflow Scheduling is considered and similarly, a membrane-based workflow scheduling algorithm, based on membrane model, is proposed. It is also implemented and the results are analysed.

Cloud computing is a convenient way to get an on-demand access to a shared pool of computing resources. These resources are configurable. As a technology, this has become very popular over the past decade. There are several definitions of cloud computing, and two most accepted definitions are by National Institute of Standards and Technology (NIST), USA and one business oriented definition given (Buyya et al., 2009). The NIST definition states that, there are five properties which decide the reason for a technology being cloud or not. There are five essential characteristics of cloud (Mell et al., 2011):

- On Demand Service

- Broad Network Access
- Resource Pooling
- Rapid Elasticity
- Measured Service

In simple terms it follows pay as you go model where the user has to pay only for what is being calculated in standard units of time. In this era internet is considered as a utility and it is perennial technology that is being used by many users all over the world. From technological perspective and business perspective, cloud computing considers internet as its base. The main motivation behind cloud computing is to allow users to use any type of technology on lease basis so that anyone can use this with minimum requirements of a browser and internet connection on the users side. Cloud computing primarily has three important categories based on its service type, these are called as service models:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

IaaS model provides computing infrastructure to the user, usually as Virtual Machine (VM). VMs of different sizes are considered and based on the users requirement these are chosen by the user. Often these services are available according to some predefined, fixed sizes, which depend on the need of the user (its application). The primary categories include Nano, Micro, Memory intensive, Compute intensive etc.; and the user selects a VM according to the application. The next category is PaaS, where 'platform' is given as a service. The platform here is any module that allows users to create application using platforms, like Google App Engine. SaaS is a service model, where software is given as a service to the users. Users can use an individual copy of the software on payment basis (per unit time). An important property of these types of systems is multi-tenancy. Multi-tenancy is a property by which multiple instances of an application with shared resources are given as service to the user. The instances are independent functional units

with primarily hardware resources being shared and the sharing details are abstracted from the user (Mell et al. (2011)).

This study concentrates on VMs which are given as part of IaaS cloud. The organizations offering the service are termed as Cloud Service Providers (CSPs), also specifically referred to as IaaS provider. There are several CSPs all over the world. The number of these providers goes beyond 100 and every cloud provider is considered to have a number of varied services ranging from 4 - 20. Overall there are more than 1.5k services available, to be chosen from. The choice of service selection usually takes place through a service broker, who is an intermediate party, between the service user and provider. Though a service broker performs many tasks, their primary job is to allow the users to select the services. All the properties collectively allow quantification of service features, which gives rise to the use of specific parameters for service evaluation. This is internally done based on the parameters which are called Quality of Service (QoS) parameters. QoS parameters are the essential indicators primarily responsible for facilitating service selection. Cloud Service Selection often involves ranking the given services, based on these QoS values. Thus, the service with the most optimal QoS values is considered as the best. There are many methods for selecting the optimal solution for selection, these are analysed in literature review and thereby research gaps are identified.

Workflow Scheduling in cloud is another part of the cloud computing technology where tremendous research is going on. Workflows are a set of interdependent tasks that have usually one entry point and one exit point. A workflow is aimed at doing a single big task collectively divided by interdependent smaller tasks. The process of mapping these tasks of workflow into different processing units (VMs) is called workflow scheduling in cloud. There has been a lot of work in this area and these have been analysed in section 2.1.

The aim is to research whether any serial or parallel bio-inspired models are available for solving these above mentioned problems and identify the gaps in the literature.

1.1 Gaps Identified in Existing Literature

After analysing the literature (in section 2). There are some research gaps that have been identified as follows:

- There are no inherently parallel model based solutions available for cloud service selection process
- There are no natural computing models used for this problem
- There has not been any significant applications of P System for cloud service selection
- There are many tools for Natural Computing but there exists no tool that supports multi-membrane system execution and no user-friendly Python based parallel simulator, which can be used for service selection is available
- There are no inherently parallel natural computing based models for workflows scheduling in cloud

To effectively tackle these research gaps, Membrane Computing-based solutions and tools are proposed for service selection in cloud and workflow scheduling in cloud.

1.2 Problem Statement and Objectives

The primary aim of this project is to Design and Develop Parallel methods for Cloud Service Selection and Cloud Workflow Scheduling. The study is divided into two major components. The first part of this work is to primarily solve the problem of cloud service selection. In literature, there are several approaches that have been proposed for service selection in cloud but there are only few service selection algorithms that are inherently parallel. The aim of this work is to design a service selection algorithm for cloud, which is efficient. To be specific there are several parameters that have to be considered for service selection. These are generally QoS parameters that have a significant impact on the service selection. Based on these parameter and using a suitable method, the services are being selected.

The second part concentrates on solving the problem of Workflow Scheduling, that can be termed as aggregate service execution. In simple terms, as an extension of the previous problem, it consists of several services which work in a group as an aggregate and these aggregate services work in coordination with each other (Workflow). This approach concentrates on scheduling these services with inter-dependencies. As the number of services increases the scheduling problem can be far more complex than the service selection problem. Thus, better mechanisms are required for both the problems.

There are three Research Objectives listed as follows:

Research Objective 1 (RO 1):

To Design a Parallel Service Selection Mechanism using P System

Research Objective 2 (RO 2):

To Implement the Designed Parallel Service Selection Mechanism for a Cloud

Research Objective 3 (RO 3):

To Design and Develop a Parallel Workflow Scheduling Mechanism using P System for Cloud

1.2.1 Objective 1: To Design Parallel Service Selection Mechanism based on P System

The objective is to develop a parallel service selection model. The service selection model developed, is based on Membrane Computing. This is a unconventional and natural computing model which is significantly different from conventional computing models. Membrane Computing model is realised using P System. The conventional model is sequential in nature whereas P System is maximally parallelizable model.

There are several variants of P System available. Primarily there are two classifications; Symbol based P Systems and Numerical based P System. The symbol based P Systems include all the P System models that were proposed in the initial phases, when this paradigm came into existence. This involves classical definition of P System, where different problems have been solved by using multi-sets as inputs and outputs. Later in 2006, a variant

of P System was proposed by Păun and Păun (2006) called as Numerical P System (NPS). This is a different kind of variant which deals with numerical values instead of multi-sets or strings in some cases. NPS structure retains the property of maximal parallelism and it has been extensively popular for numerical problems. Another version of NPS is called as Enzymatic Numerical P System (ENPS) and has been proposed by Pavel et al. (2012). It is better than NPS as it offers additional control over the basic components of membrane. In this study, ENPS is used for solving the Service Selection problem.

As part of the first objective, methods related to generic service selection are proposed. The first method (ENPS-IAHP) is primarily proposed for weight calculation of the attributes and the second method is for service selection (alternative selection) for a generic Multi-Criteria Decision Making (MCDM) scenario (ENPS-IPROMETHEE).

1.2.2 Research Objective 2: To Implement the Designed Parallel Service Selection Mechanism for a Cloud

The second objective is one of the primary contributions of this thesis. Here a couple of method for service selection in cloud are proposed. These methods are based on amalgamation of MCDM method (particularly ITOPSIS) and ENPS. These methods are specifically designed for cloud based services. Two robust methods for cloud service ranking (selection) have been developed and implemented. The results are compared with standard methods and the proposed approach is found to be robust.

To realise the capability of the proposed ENPS approaches, tools that can completely simulate the existing work are needed. Therefore, two useful tools are developed, that have been discussed in the coming sections. These developed tools have later been used for realizing the proposed methods. The first tool is for multi-membrane system execution; as the problem which is dealt with involves multi-membrane systems and a generic tool is required for executing the proposed models. The second tool (GPUPeP) is a GPU based tool which simulates the exact ENPS model with complete parallelization using Graphical Processing Units (GPUs).

1.2.3 Research Objective 3: To Design and Develop a Parallel Workflow Scheduling Mechanism using P System for Cloud

The third objective is to Design and Develop a Cloud based workflow scheduling algorithm for efficient workflow scheduling considering reduced makespan. A workflow contains many tasks which are dependent on each other. Scheduling workflows is a NP-Complete problem, and there is no polynomial time algorithm to perform workflow scheduling accurately. A method based on the amalgamation of a Heuristic technique and inherently parallel membrane-based ENPS is proposed. The proposed model involves a multi-membrane system that is used for generating the sequence of schedule and subsequently makespan is calculated which is compared with standard approaches.

1.3 Research Methodology

A quantitative research methodology has been followed to analyze the results obtained during this research work. In the initial stage, a critical review is conducted on the related research works. The research methodology consists of five phases which are described as follows:

Phase 1: This consists of critical review of all the approaches coming under the purview of the project with a gap analysis at the end. A critical review has been done according to specific areas of research and after analysing the related works, a review is written. This phase is required, to know about the existing research works present in this area and to further carry out an accurate gap analysis which facilitates proceeding in the correct direction.

Phase 2: This phase involves the first part of design and implementation of the proposed solution using membrane computing paradigm. Specifically, Numerical P System (NPS) variant called as Enzymatic Numerical P System (ENPS) is used, as the model is suitable for the problem that is intended to be solved. Initially, a method that allows calculation of weight is considered (Analytic Hierarchy Process). Further, a tool is developed which allows multiple membrane systems to be used together as a single entity.

Phase 3: This phase involves design and development of first service selection method.

An outranking based method named as ENPS-IPROMETHEE is developed. This is a single membrane system based method. The method is designed and used with PeP, Python-based tool for realizing ENPS. The proposed method allows combining the output of ENPS-IAHP, if required or direct assignment of weight is allowed. After the attribute weights and details of the alternatives are passed, the membrane based ENPS-IPROMETHEE gives the output. The results are compared and analysed, by its structure this method is inherently parallel and it proves to be sensitive in nature.

Phase 4: This phase involves design and implementation of robust membrane-based method for cloud service selection. The work proposes ENPS-ITOPSIS, where the weight and alternative corresponding weight attribute is considered as input and the output obtained is ranklist of the given alternatives. As this method is an inherently parallel model, GPU based tool called GPUPeP is developed to implement membrane system. The tool only works for Enzymatic Numerical P System. The results show that the tool is better than the existing tools available. This tool is used for simulating multi-membrane system model called ENPS-ITOPSIS in a parallel manner.

Phase 5: A Workflow Scheduling method in cloud based on ENPS model is designed and implemented. This workflow based ENPS mode primarily gives the sequence of schedule based on which the the makespan is obtained. The results show that the proposed membrane-based model is better than other few standard approaches.

1.4 Thesis Contributions

Few contributions of the thesis have been listed below:

- **Membrane-based Weight Determination method**: A membrane-based attribute weight calculation method based on ENPS and IAHP is designed and developed.
- **Membrane-based MCDM method**: A membrane-based MCDM method is proposed (ENPS-IPROMETHEE) for selecting best alternatives. The results show the algorithm to be sensitive to weight changes.
- **Multi-membrane System Execution Tool**: A multi-membrane system execution

tool using Python has been developed. The tool is one of its kind, that allows to combine several membrane systems. Additionally it allows inter-conversion of XML based membrane system to PeP based membrane systems and vice-versa.

- **Membrane-based Cloud Service Selection Method**: ENPS based service selection methods, for cloud based services is proposed. The proposed methods are specifically designed for cloud service selection and thus the results indicate it to be robust when compared with standard methods.

- **GPUPeP: GPU and Python based tool for ENPS**: GPUPeP is a tool based on Python and CUDA, that allows to simulate ENPS models. The results have been analysed with different case studies and these tools prove to be significantly better than the other contemporary tools.

- **Membrane-based Workflow Scheduling in Cloud**: Membrane-based workflow scheduling for cloud is proposed. The method is based on ENPS where it is used for generating the schedule, and based on which the makespan is calculated and it is compared with other contemporary standard methods. The results show proposed algorithm to be better.

1.5 Thesis Structure

The rest of the thesis is organized as follows:

- Chapter 2 presents an introduction and overview of recent works in the area of cloud service selection, Membrane Computing Application, Membrane Computing Tools available and related workflow scheduling algorithms

- Chapter 3 (RO 1 and RO 2) presents the membrane based weight calculation method called ENPS-IAHP, further this may be used for the sensitive method ENPS- IPROMETHEE that has been proposed in this chapter. Both the methods are elaborated and its results are analysed.

- Chapter 4 (RO 2) is completely devoted to the stand alone tools that have been developed as part of this work. The first tool is for executing multiple membrane

systems together additionally they allow file inter-conversion between two standard ENPS description formats. The second tool is a simulator for ENPS and is based on Python and uses GPUs at the back end.

- Chapter 5 (RO 1 and RO 2) elaborates on the proposed method of service selection for cloud based services named ENPS-ITOPSIS. This is an inherently parallel robust method. The results have also been analysed and compared in detail.

- Chapter 6 (RO 3) is about membrane-based method for workflow scheduling in cloud. This method is ENPS based and generates proper sequence of the schedule, based on which scheduling is done. The resultant makespan is compared with the standard methods and the proposed method proves to be better than the others.

- Chapter 7 presents the thesis summary and conclusions of the research work. This chapter also highlights some of possible future research works.

Chapter 2

LITERATURE REVIEW

This chapter presents a literature review on several important topics that are part of the problem being solved. Cloud Service Methods, particularly Multi-criteria Decision Making (MCDM) Models, for Cloud Service Selection are considered. Research Gaps are identified. Diverse Applications of Membrane Computing Models, tools used for realizing Membrane Computing Models and different suitable P System models available for solving the current problem are discussed. Further, a discussion on methods available for cloud workflow scheduling is presented.

2.1 Cloud Computing

As per the definition of National Institute of Standards and Technology (Mell et al., 2011), cloud computing is a model which is available anytime and anywhere i.e., it can be used any time; whenever needed, for accessing the resources (Hardware) and any part of the world, provided that there is internet connection. This cloud model consists of essential characteristics like on-demand self service, broad network access, resource pooling, rapid elasticity and measured service (Mell et al., 2011). There are three basic service models, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). There are primarily four deployment models namely Private Cloud, Public Cloud, Hybrid Cloud and Community Cloud.

IaaS is considered as the base for all the contributions made in the thesis, and it is

related to cloud infrastructure where a cloud service refers to Virtual Machines (VMs). There are several technologies in cloud that are essential for a cloud to work, out of which virtualization is one of the most important (Mell et al., 2011). Virtualization allows the cloud to create several virtual machines on a single physical machine. The created VMs act as different machines and perform the computation as separate physical machines would do.

The primary contributions of the thesis include Cloud Service Selection approach based on membrane model, tools assisting the implementation and workflow scheduling using membrane model. Thus, the specific model that is proposed is obtained after a thorough literature review. Reviewing of literature is done in a certain pattern; the first component is about cloud service selection and the algorithms related to cloud service selection. The second module of literature deals with the membrane computing models that are available and also the membrane tools available as part of literature. Using this the best tools are selected along with the best membrane models suitable for the problem. Further, second logical part of the thesis involves workflow scheduling in cloud; in the literature, workflow related scheduling algorithms with its classifications are discussed.

2.2 Service Selection in Cloud

Cloud Computing is a decade old technology which has changed the landscape of the internet-based business model (Buyya et al., 2009). This technology surfaced unheralded, a decade ago and has been growing since. It now stands with several inherent complex problems, as a result of its expansion. Out of several issues being researched, service selection is one of the prime issues which is getting special attention. Service selection is a process of selecting one, from a pool of available services often based on multiple Quality of Service (QoS) attributes. These QoS attributes have often tested the favorability of the services to the user, based on the requirement and aim of the project. There are three components of a normal cloud service selection model; the service consumers, the broker and service providers (Sundareswaran et al., 2012). A service consumer is a user who leases services from the service providers. The broker acts as an interface between the service consumer and service provider. Brokers allows the consumers to access the service

providers. Several service providers are available and the broker handles communication with a set of service providers. This is the most appropriate place for service selection algorithm to be present as in 2.1.

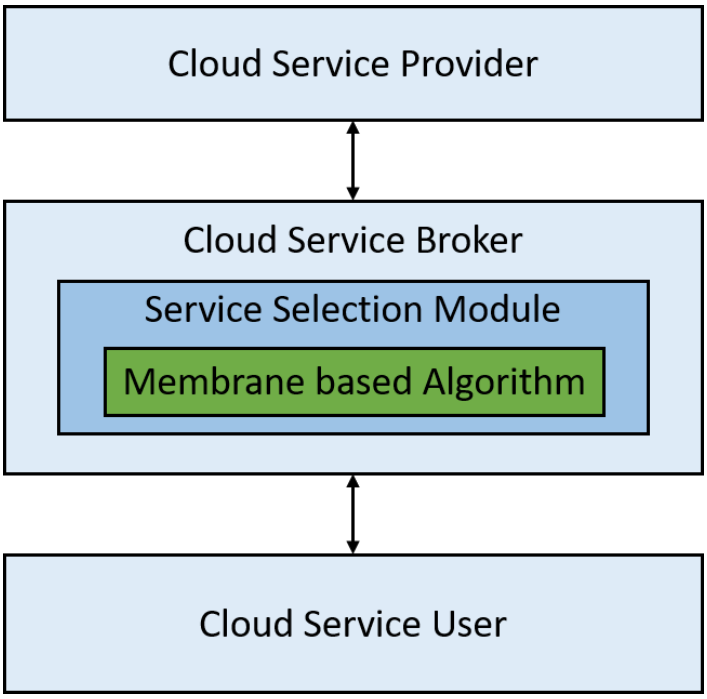

Figure 2.1 Cloud Service Selection

Considerable work has been done in this area and each work has a specific advantage. A Literature Review has been done and some important works are discussed in this section. Almost all of the works, except a few, consider sequential model for designing the algorithm whose complexity increases non-linearly for linear increase in the number of services. Considering this trend, a parallel natural computing model, for service selection in cloud is proposed.

There are several studies conducted in the area of cloud service selection. According to Sun et al. (2014), there are primarily four classification methods for cloud computing; MCDM based approaches, Optimization based approaches, Logic-based approaches and other approaches. Out of the four approaches the most popular one is Multi-Criteria based decision making approach. The paper by Sun et al. (2014) gives a detailed account of these approaches. Further after 2014, there have been several works in the area of cloud service selection, specifically using MCDM methods. Discussed further are some important works

for service selection in cloud and these primarily use MCDM methods.

Garg et al. (2011) have developed a framework for service selection for cloud computing. This is one of the first attempts to provide a framework for cloud service selection that can assist the user to select the Cloud services. The framework uses Analytic Hierarchy Process (AHP) as the base method for ranking the services in order, according the service attribute values.

The service attributes are according to Service Measurement Index (SMI); though not a method, SMI is an important work in this area of cloud service selection. It has been proposed by Siegel and Perdue (2012) as Service Measurement Index, based on CSMIC (Cloud Services Measurement Initiative Consortium) directives. It is an elaborate framework, that gives the details of services, that can be measured and used by providing a hierarchical structure of properly classified cloud service attributes. It is one of the few attempts to standardize the criteria for cloud service selection and has been used as a standard in several works (Garg et al., 2011; Somu et al., 2017).

On similar lines, another method by Lang et al. (2018) tries to give a set of services that can be used for service selection in cloud. They have several considerations like company sizes, cloud service models, industries and a few others. The criteria is obtained after applying a Delphi study over these QoS attributes.

Several other approaches use MCDM methods for service selection in cloud. Lee and Seo (2016) have proposed a Hybrid MCDM model for cloud service selection, using the combination of Balanced scorecard, Fuzzy Delphi method and Fuzzy Analytic Hierarchy Process Method (FAHP). The applied methods are better than other methods for several reasons, one of the primary reason being, reduction in vagueness because of human decisions.

Kumar et al. (2017) have proposed a solution for service selection in cloud using a MCDM approach interlaced with fuzzy approach. In this study, TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) has specifically been used for Decision Making here. The fuzzy approach aims at capturing the uncertain aspects of attributes in

order to give proper result. The proposed approach can be used for objective as well as subjective attributes. The approach has finally been tested using cloudharmony dataset (Read, 2014) and it is compared with other MCDM Methods.

Patiniotakis et al. (2015) propose a cloud service selection method for service brokers, considering preferences, called as preference based cloud service recommendor system (PuLSaR). It has used fuzzy set theory to handle vagueness of the service metrics and it is designed to be used with linguistically expressive values.

Abdel-Basset et al. (2018) propose a framework for evaluating cloud computing services. Here Neutrosophic Analytic Hierarchy Process (NAHP) is used for service selection. NAHP allows calculation of weights given the user preferences, it also allows consistency check, to check the proper assignment of weight calculation, thereby assuring correct assignment of weights. Neutrosophy reduces vagueness involved in criteria. Nawaz et al. (2018) propose a markov chain based MCDM approach cloud service selection. Markov chain is used to capture and model uncertainty in the changes and Best Worst selection is used for checking through changing user selection. The results proved to be more consistent than other AHP based approaches.

Sun et al. (2016) have proposed a hybrid fuzzy framework for cloud service selection called Cloud-Fuser. A hybrid algorithm based on Fuzzy ontology, Fuzzy TOPSIS and Fuzzy AHP are considered. One of the points of novelty includes usage of fuzzy ontology which is applied for service matching. The proposed method is compared with eight other TOPSIS variants and the proposed approach performs better than other approaches. Kaveri et al. (2017) use entropy based fuzzy method for MCDM service ranking approach. The work proposes a method called as E-FPROMETHEE, this method is an integration of fuzzy method and PROMETHEE method. Cloudharmony based dataset has been used for the case-study. The dataset has been tested for trustworthiness, untrustworthiness and uncertainty.

Yadav and Goraya (2018) proposed a two way ranking based service mapping for cloud. The two way evaluation of service consumers as well as service providers, is considered. The authors use AHP for calculating the rank in both the cases and then try mapping

both the rankings. Sensitivity analysis is made to check the robustness of the proposed algorithm. The authors implement the proposed work using Cloudsim (Calheiros et al., 2011) and the execution time of the service mapping algorithm proves to be better.

Al-Faifi et al. (2019) propose a hybrid approach for cloud service selection. The hybrid approach is on MCDM algorithm. Three methods for doing so, namely, DAMETAL, K-Means and Analytic Network Processing (ANP). A realtime dataset collected over a year from Saudi Financial Department (Al-Faifi et al., 2018) is used. Ma et al. (2016) propose a trust based cloud service selection method; a MCDM algorithm, which is time-aware and is based on interval neutrosophic set, with the method named as Cloud Service Interval Neutrosophic Set.

Soltani et al. (2018) propose a QuaRAM (QoS Aware Cloud Application Management) service recommender system; it is a case based reasoning hybrid system. This uses case based reasoning with MCDM algorithm, specifically TOPSIS for service selection. They also propose a Virtual Machine (VM) consolidation method after analysing resource utilization. Further, feedback is monitored and reinforcement learning is used for improving the system.

Ur Rehman et al. (2014) propose a parallel service selection for cloud, based on QoS history. The authors use a customized parallel multi-criteria based decision making algorithm. This is one of the first attempts which uses parallel service selection in cloud.

There are few methods which don't use MCDM Methods. The one developed by Somu et al. (2018) uses an optimization technique, Bio-inspired method, called as Binary fruit fly algorithm. They have developed a trust-centric optimal ranking approach. Unlike other algorithms this is a optimization technique used for cloud service selection. There are two important concepts namely; hypergraph and fruit fly optimization algorithm, which, when combined together, give a proper result. The algorithm proves to give an optimal trustworthy service selection with notable precision and stability. There are also other similar applications of Bio-Inspired methods in cloud service selection and cloud.

Ghosh et al. (2014) propose a framework for easier cloud service selection. This frame-

work is based on an overall perceived interaction risk which also establishes a relationship between them, trustworthiness and competence of service. Finally a risk model based on the three factors mentioned above is proposed.

Somu et al. (2017) propose a Hyper-Graph based Computational Model (HGCM) for cloud service selection. HGCM uses an hyper-graph based technique namely; Minimum distance-Helly Property for service provider evaluation. Here the SMI model has been used for evaluation.

Askarnejad et al. (2018) have proposed a cloud service selection algorithm for peer-assisted environment, which is network and application aware. The method proposed, Peer Cloud Assisted (PCA), allows service selection in a hybrid cloud environment. The method uses greedy algorithms, B+ trees and Set theory as its ingredients to obtain proper results for cloud services.

Based on the literature review of service selection, it is found that,

- Very few instances of parallel cloud service selection are available
- There are almost no natural computing models used for this problem
- MCDM methods have been extensively used for cloud service selection
- There have been many MCDM methods that have been used, out of which AHP and TOPSIS seems to be more prevalent methods that have been used.

The aim is to propose a parallel, robust model for solving service selection problem (ranking the services). To create a parallelized model, we aim to choose an inherently parallel model that can be combined with MCDM properties to obtain a new, wholly parallel and robust approach specific to this problem. MCDM based models are chosen as the base solution because of its structure and proved competence in the area of cloud service selection.

After studying the service selection problem for cloud and the related literature, it is observed that the process of service selection can be parallelized. Thus, a suitable parallel

model can be applied to this process. Among the methods mentioned in the literature, it is found that certain MCDM methods which are predominantly used for service selection (AHP, TOPSIS), can be parallelized and are usually robust in nature. Parallelization can be done by applying a universal model. According to the requirement, there are several inherently parallel computing models available like Molecular Computing (DNA Computing, Membrane Computing, Peptide Computing), Amorphous Computing, Quantum Computing etc. (Kari and Rozenberg, 2008). Primarily the Computing based paradigms have been considered for our solution as, unlike the other serial technology-based parallel models, these models are computing paradigms that are inherently parallel.

Among the reviewed models, particularly membrane computing is chosen because it is an inherently parallel model, simple architecture, many variants with most of them being Turing universal, has several applications noted as per the literature and one of the important reasons is the structural suitability with the problem being solved. Thus, Membrane-based parallel methods are proposed for cloud service selection. In recent years there have been several direct applications of membrane computing as membrane algorithms.

A membrane computing model realised as P System is considered. Membrane Computing is a natural computing paradigm whose idea has been conceived by Gheorghe Paun in 1998 (Dassow and Păun, 1999). In his seminal paper in 2000, he has given the details of Membrane Computing (Păun, 2000). P System is inspired by the nature, in particular, by a living cell. This is a natural computing paradigm, where complete conventional model is being restructured as a membrane. A membrane model has a hierarchical structure of membranes, where the internal components are disjoint. The outer membrane is called as skin membrane and all the other membranes are contained inside it. There can be any number of membranes present inside the skin and all these membranes can communicate with each other, i.e. they can pass information between them. This structure inherently supports parallelism, based on which the whole design works. Unlike other parallel models which have a limited scope or constrained scope of application, P System has a wider range of application owing to being a computational paradigm. Further, description about the model is given in Păun (2000); Paun et al. (2010). There are numerous variants of P Systems available (Paun et al., 2010). Each variant has a different structure and is de-

signed for a different purpose. One such specific variant called Enzymatic Numerical P System (ENPS) is used for the chosen problem of Multi- criteria Decision Making (for cloud service selection).

2.3 Membrane Computing

Membrane Computing is a natural computing based paradigm which has been proposed by Gheorghe Paun (Păun, 2000). The device used for realising membrane computing is called P System. The first model has been proposed in 1998 and further elaborated in 2000. The basic structure of membrane is as given in figure 2.2. Unlike the conventional paradigm where the base is a sequential, here the base is completely parallel.

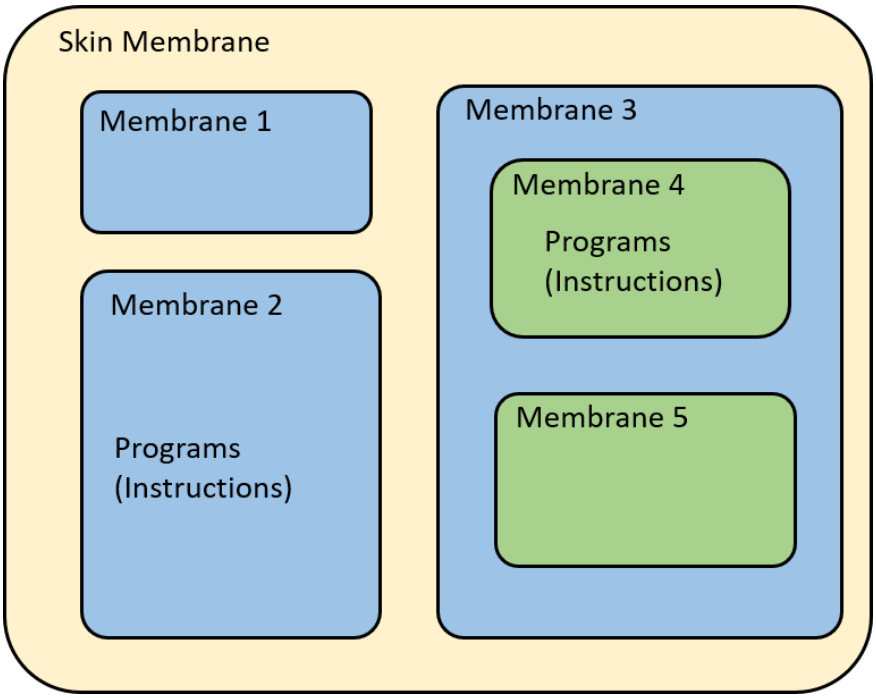

Figure 2.2 Membrane Structure

The important structural elements are as follows:

Membranes: Membranes are defined compartments defined inside which there are re-writable rules. There are two important types of membranes called as Skin Membranes and Other membranes (Child Membranes). Skin Membrane is the outermost membrane with one mandatory skin membrane. The membranes have a hierarchical structure, with

the outermost membrane (skin membrane) being upper most in the hierarchy. A skin membrane can have any number of child membranes.

Rules: Every membrane consists of several rules which are called as evolution rules. These rules are multi-set based re-writable rules. The base model uses multi-sets and there are other subsequent models that use strings and numerical values as the basic unit of operation. Primarily however, only multi-sets are used for evolution rules. Each P System works in a different manner, the only thing in common being parallelism. There are several membrane-based algorithms which throw light on various applications of membrane computing.

2.3.1 Membrane-based Algorithms

Membrane computing paradigm can be effectively applied to multiple problems, as membrane algorithms. There are several applications of membrane-based algorithms and every variant of the P System (Membrane Computing) has this property, which is usually the primary reason for going with membrane algorithms. Membrane-based algorithms are either loosely or strictly based on the membrane computing paradigm.

Several works have been carried out using membrane algorithms; one of the firsts is image thresholding (Peng et al., 2012, 2013). As an advanced application, Wang et al. (2012) have also proposed a P System based image segmentation technique based on image thresholding. Later Huang and Wang (2006) developed membrane-based optimization algorithms for single-objective optimization.

Clustering is an important area that has been looked into by P System researchers. There have been several works in this area. The first of its kind is by Peng et al. (2014), where a simple and effective clustering algorithm using membrane computing has been designed. Similarly, there have been several other works that have come with this particular application, where a membrane-based structure is primarily used to enhance the efficiency of clustering approaches (Peng et al., 2015, 2016, 2017; Hu et al., 2017).

Another promising series of applications of membrane algorithms is for evolutionary computing, called as evolutionary membrane algorithms, which have been discussed in

detail in a paper by Zhang et al. (2014b). Apart from this, there is another application of membrane computing for swarm-based optimization techniques. In specific, there have been few works on enhancing Particle Swarm Optimization (PSO), by using membrane-based algorithms. New Membrane-based PSO approaches give improved results and are compared with other approaches, that prove to be better (Singh et al., 2014; Wang et al., 2015).

Another important application of the membrane-based algorithms has been the sudoku solver. In 2010 Díaz-Pernil et al. (2010) proposed a membrane-based algorithm for solving the sudoku problem. Several other researchers have further considered this problem. The latest work by Singh and Deep (2016) is one of the efficient methods of solving the sudoku problem, where the approach solves easy and medium problems with considerable ease, giving good efficiency, compared to other similar sudoku solvers. The applications of the membrane-based algorithms, as discussed in this section, shows its power, and in general, allows analyzing its suitability to different problems. It is inferred that these kinds of algorithms are suitable for complex problems, where parallelism can be infused.

To choose an appropriate membrane computing variant, apart from its applications, several membrane computing variants' structure and properties have been analyzed. There are few primary classifications based on the structure; namely, Cell-based P System, Tissue-based P System and Neural P System (Paun et al., 2010). Among these, the most structurally and functionally suitable to the problem is Cell-based P System. Cell-based P Systems have a cell (Animal cell) like structure. All the membranes are placed inside a skin membrane in the hierarchical order, consisting of parent membranes and child membranes. Within the Cell like membrane system there are few variants like Active membranes with division rules, Active membranes with creation rules, Transition P systems, Symport/Antiport P systems, Stochastic P systems, Probabilistic P systems and Numerical P System etc. (Paun et al., 2010; García-Quismondo et al., 2009).

The variants of cell like P System can be further classified into two categories, Symbol based P Systems and Numerical based P System. Most of these variants that have been built upon this definition are based on multi-sets and strings. Though almost all of these

models are universal in nature, but finding a solution for a generic sequential problem with this setup is a difficult task. The difficulty is not attributed to the use of multi-sets or the membrane structure but its translation to solve the problem, which are primarily based on numbers. Thus a P System Model that allows numbers to be used directly can be beneficial.

Considering this, in 2006, Gheorghe Paun et al. have proposed a number based Numerical P Systems inspired by Economics (Păun and Păun, 2006). This is a different kind of variant that deals with numerical values, instead of multi-sets. NPS structure has retained the property of maximal parallelism, and it has become extensively popular for mathematical problems. This model is found to be suitable for this research. However, there is a problem with this model; that of uncontrollable programs and a single program per membrane. Here, programs are the basic rules (instructions) in the membranes that are responsible for computation.

Subsequently, this problem has been solved by Pavel et al. (2010), where they have introduced the use of multiple programs for a single membrane and provide a variable, called an enzyme to control the execution of programs. This is called as Enzymatic Numerical P System. Further, to support the hypothesis, some work specifically use ENPS to solve the problem. The first work to apply ENPS to a problem is by Pavel et al. (2012), who have used it for robot localization problem. This is the first practical application of ENPS. Earlier, ENPS has been used explicitly for the Pole balancing problem by Llorente Rivera and Gutiérrez Naranjo (2015). Further, ENPS has been found to be useful in its application for robot controllers (Zhang et al., 2017a), where it has been used for the design and implementation of robot controllers. Later, a robot path planning algorithm, based on Rapidly-exploring Random Trees (RRT), using ENPS has been proposed by Pérez-Hurtado et al. (2018). An ENPS simulator specific to this problem has been designed to generate ENPS based RRT trees for path exploration; thus, eventually, path planning is done. Further, an enhanced method for the said approach is used for RRT and RRT* algorithm (Pérez-Hurtado et al.). This latest work is one of the best works in this area, which in addition to the model, proposes a new specific simulator for the RRT algorithm using Membrane Computing. Another addition to this is the work by Wang et al., who uses this for a multi-behavior co-ordination controller design for the robot. This work is inspired by

the previous works in this area (robots).

If we carefully look at these works, these signify two things; all the works support parallelism, and these involve real values (numbers) for operation. Thus, researchers have successfully used ENPS, and in many cases have designed it with a specific simulator for the problem. Taking a cue from these works, which strictly implement ENPS for some significant real-time problems, this research proposes a similar ENPS based solution for our problem.

2.4 ENPS Structure

An important model for numerical based P System is Numerical P System (NPS) (Păun and Păun, 2006; Pavel et al., 2010). NPS is an off shoot of P System that has been envisioned by Păun (2000). This model has the same membrane structure but the rules used inside each membrane are called as programs and they use numerical values.

Further, there is a variant of Numerical P System and it is called as Enzymatic Numerical P System (ENPS). The ENPS has one more variable added to the usual Numerical P System which is called as Enzyme. The basic advantage of using ENPS over classical Numerical P System is as follows (Pavel et al., 2010):

- **More than one production function per membrane.**
- **Control over the execution of the rules in each membrane using the enzymes.**

The structure of NPS is defined as follows:

$$\prod = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0))) \tag{2.1}$$

- H is an alphabet with m symbols (used as labels of membranes),

- m is the number of membranes used in the system, m $\geq$ 1,

- μ is the membrane structure with m membranes.

- Var_1 to Var_m are the set of variables that are available.

- $Var_1(0)$ to $Var_m(0)$ denotes the initial value of first to last variable, in sequence, that have been defined.

- Pr_1 to Pr_m denote the programs that are available in the whole membrane system.

There are two types of programs available for ENPS, one is normal program, as in NPS given in equation 2.2) (Pavel et al., 2010; Pavel and Buiu, 2012; Leporati et al., 2013),

for a given region i, $x_{l,i},..,x_{k_i,i}$ be some variables from Var_i and

let $F_{l,i}(x_{l,i},...,x_{k_i,i})$ of a given program $P_{l,i} \in Pr_i$ and $c_{l,1},..,c_{l,n}$ be considered as positive integers. Leporati et al. (2013)

The programs that have been defined here consist of two parts: a production function and a re-partition protocol. The production function is like a normal function that uses variables that have been defined (in that membrane). These production functions are similar to normal functions, which calculate and give certain output. The only condition being that the variables used here should be defined within the region of the specific membrane under which these production functions are also defined.

The next part is called a re-partition protocol; this re-partition protocol consists of variables that can store the values sent by the production function. This, inadvertently, is like passing the values of the functions to the variables that may or may not be inside the available same region. The re-partition protocol may consist of certain values that may be defined within other membrane regions, but these variables have to be a part of the main membrane system i.e., must be defined in any of the other membranes.

The re-partition protocol consists of two parts again; one is the distribution fraction (weight) part, which assigns equal division of weights to the corresponding variables. Here each variable in re-partition protocol is assigned $\mathbb{Z}_{\geq}1$. The value passed on to the production function is divided according to the value assigned to the variable proportionately.

$$F_{l,i}(x_{l,i},...,x_{k_i,i}) \;\rightarrow\; c_{l,1} \mid v_1 + c_{l,2} \mid v_2 + \ldots + c_{l,n_i} \mid v_{n_i} \tag{2.2}$$

and the next program, mentioned in 2.3 is for ENPS,

$$F_{l,i}(x_{l,i},...,x_{k_i,i}) \mid_{e_{j,i}} \;\rightarrow\; c_{l,1} \mid v_1 + c_{l,2} \mid v_2 + \ldots + c_{l,n_i} \mid v_{n_i} \tag{2.3}$$

All the programs that have been defined in equation 2.2 and equation 2.3 are based on cycles or steps i.e. execution at time t. After each iteration or step (time unit t), the values used in production function are deemed to be consumed, and the values are reset to 0 for the next iteration $(t+1)$; and the variable which is not used in production function retains its original value available.

The primary condition to operate using enzymes is that the value of at least one assigned variable must always be lesser than the enzyme. In other words, the value of the enzyme $(e_{j,i})$ has to be greater than the smallest variable available in the production function for the program to be true.

If the value of the enzyme that is being applied is lesser than the smallest variable available, then the condition may prove to be false, and the production function tends to be inactive. The production functions can be indirectly selected or controlled using the values of the enzymes. As mentioned earlier, based on the value of the enzyme and the variable, the function can be active or inactive. This lets the enzymes be used with the NPS structure, which may allow solving many problems that need the production function to be selected. Thus it depends on the developer to develop a system that can utilize the power of enzymes and apply ENPS.

Further, to choose a proper tool to implement the membrane this study has analysed membrane based tools in detail.

2.5 A Review of Tools Available for Membrane Computing

There are several applications of Membrane Computing. P Systems, the device used to realise membranes are characterized by high parallelizability. It is bio-inspired computing paradigm which has a lot of applications because of its inherent structure. It is inspired by structure and functioning of a living cell. As the concept is based on the living cell, it is also being seen as a tool to be used to emulate or describe biological processes, which is one of the important applications of P Systems; this may even revolutionize the way biological processes are studied.

To realize the power of membrane computing it is necessary to develop the tools that

will emulate/ simulate biological processes. There is also a need to have P System tools that allows the simulation of P System to test / realize its computational properties and mathematical properties. Both types of the tools are necessary. The number of tools for the latter is more compared to the former. Initially, there were several tools developed with the sole purpose of demonstrating the power of P Systems for solving computational and Mathematical problem and then gradually tools that are being used for several applications of membrane computing, have been developed.

These tools, apart from testing the P Systems, also allow the user to use P System to visualize and understand the way P Systems work; thereby giving more clarity to the user on certain fine issues of P Systems. Most of the systems are born out of research and are created for immediate necessities / requirement for the researchers. As the membrane computing paradigm has evolved theoretically, there are several methods and mechanisms to experience the practical implication of membrane computing. These often come as simulators.

The simulators or simulation tools can primarily be classified into two, the tools that concentrate more on biological aspects of membrane computing and help use the membrane computing paradigm for simulating biological processes; and, the set of tools/ simulators that have been developed for using the membrane computing paradigm for solving problems related to Mathematics and Computer Science. All the simulators in these two areas have been discussed in this paper.

There are several tools and softwares that have been developed by different researchers working on membrane computing. A list of these softwares has been given in the website that has been maintained for P Systems (RGNC, 2016). There are several initial studies made on tools for membrane computing. There are also a few surveys that have been done before (Paun et al., 2010; Martínez-del Amor et al., 2014; Ciobanu et al., 2006). Though a list also has been maintained (RGNC, 2016), this work is an attempt to classify all the available tools according to their topic, so that there is a systematic literature review of the tools that are available for P Systems. Apart from these, there is also some detail about all the simulators, giving a brief idea about the purpose, language and other properties of the

simulator.

As discussed, Membrane Computing is a vast topic. There are many researchers in this area working and expanding it from all directions. As the research progresses and development progresses, different types of tools are required for different areas. Thus, keeping this in view, several tools have been developed regularly according to the need of the researchers.

The tool classification is based on eventual application of P Systems i.e. computational/ general application or specific application. In this section, simulators/ tools that have been developed for P Systems are classified. The P System/ application specific tools have been listed separately and the general tools have been listed separately.

2.5.1 P System tools that are specific to a particular application or type

This section discusses the details about the tools / simulators that are specific to a type of P System or specific kind of application. Specific tools refer to the tools:

- That are designed only for a particular type of P System (Transition P Systems (Păun and Rozenberg, 2002), Numerical P Systems (Paun, 2012) etc.)
- That have a very specific type of application (eg. Generating trees (J Plant (Rivero-Gil et al., 2011)))

In this section all the tools that come under the above area are considered. The initial trend of simulators designing has been for a specific type of P Systems. It all started from designing the simplest transition P System; later there have been several system which have been designed for specific kinds of P Systems. On the whole, most number of the simulator are only for specific kinds of P Systems as in table 2.5.1.

Tool/ Software	**Name of the developers**	**Base tool /Framework /Language**	**Purpose/ Application**
Membrane Computing in Prolog (Malita, 2000)	Mihaela Malita	Prolog	Transition P System

On a LISP Implementation of a Class of P Systems (Suzuki and Tanaka, 2000)	Yasuhiro Suzuki, Hiroshi Tanaka	LISP	Transition P System
Membrane Software A P System Simulator (Ciobanu and Paraschiv, 2002; G. Ciobanu, D. Paraschiv, 2001)	G. Ciobanu, D. Paraschiv	Visual C++	Two Variants of P Systems
A CLIPS Simulator for Recognizer P Systems with Active Membranes (Pérez Jiménez and Romero Campero, 2004)	Mario de Jesus Perez Jimenez and Francisco Jose Romero Campero	CLIPS	For Recognizer P Systems with Active Membranes
A MzScheme implementation of transition P systems (Noval et al., 2002)	Delia Noval Balbontín, Mario J. Perez Jimenez, and Fernando Sancho Caparrini	MzScheme	Transition P System
A Software Simulation of Transition P Systems in Haskell (Arroyo et al., 2002)	Fernando Arroyo, et al.	Haskell	Transition P System
Distributed Simulator for Transition P Systems (Syropoulos et al., 2003)	Apostolos Syropoulos, et al.	Java (with standard UDP Protocol)	Distributed in Nature, Works for Transition P System
Sevilla Carpets (Ciobanu et al., 2003; Orellana Martín et al., 2014)	G.Ciobanu, Gh.Paun and Gh.Stefanescu	Python	Comparing solutions for subset sum problem
SubLP-Studio v0.1 (RGNC, 2016)	Alexandros Georgiou	Java	For L System and P System
A Prolog Simulator for Deterministic P Systems with Active Membranes (Cordón-Franco et al., 2004)	Andres Cordon-Franco, et al.	Prolog	Deterministic P
P Systems Running on a Cluster of Computers (Ciobanu and Wenyuan, 2003)	Gabriel Ciobanu , Guo Wenyuan	C ++, MPI	Generic

Modelling biological processes by using a probabilistic P system software (Ardelean and Cavaliere, 2003)	Ioan I. Ardelean, Matteo Cavaliere	-	For Biological processes
SimCM (Nepomuceno Chamorro, 2004)	M. Isabel Nepomuceno Chamorro	Java	Transition P System
A simulator and an evolution program for conformon-P systems (Frisco and Gibson, 2005)	Pierluigi Frisco, Ranulf T. Gibson	Java	Conformon P System
A Simulator for confluent P systems (Gutiérrez Naranjo et al., 2005)	Gutierrez Naranjo, Miguel Angel, Mario de Jesus Perez Jimenez, and Agustín Riscos Nunez	Prolog	For more than one type of P System
Simulation Software for Membrane Approximation Algorithm (Nishida, 2006)	T. Nishida	-	Specifically designed for membrane approximation algorithm
Vibrio Fischeri (RGNC, 2016)	P. Cazzaniga, D. Pescini	C	For Biological Process
Dynamical Probabilistic P Systems (Pescini et al., 2006)	P. Cazzaniga, D. Pescini	MPI and C	Probabilistic P System
Tissue Simulator: A Graphical Tool for Tissue P Systems (Borrego-Ropero et al., 2007)	Rafael Borrego–Ropero, Daniel Dıaz-Pernil, and Mario J. Perez-Jimenez	Java and C#	Specifically designed for Tissue P Systems
DasPsimulator (Das and Renz, 2006)	D. K. Das and T. Renz	Java	P System Simulation with Active Membranes for Transition P Systems

A Tool for Using the SBML Format to Represent P Systems which Model Biological Reaction Networks (Nepomuceno Chamorro et al., 2005)	Isabel Nepomuceno, Juan Antonio Nepomuceno, Francisco Jose Romero Campero	CLIPS	To represent Biological processes
A Software Tool for Dealing with Spiking Neural P Systems (Ramírez Martínez and Gutiérrez Naranjo, 2007)	Daniel Ramirez-Martinez, Miguel A. Gutierrez-Naranjo	Xbase++ and SWI – Prolog	Spiking Neural P System
MetaPlab: a virtual laboratory for modeling biological systems by MP systems (Castellini and Manca, 2008)	Alberto Castellini and Vincenzo Manca	Java	Bio-Systems
Simulation of P Systems with Active Membranes on CUDA (Cecilia et al., 2010b)	Jose M Cecilia., et al	C and C++ programming language along with CUDA Extensions	P Systems with Active Membranes
A P-Lingua based simulator for tissue P systems (Martínez-del Amor et al., 2010)	Miguel A. Martinez-del-Amor et al.	P-Lingua	Specifically designed Tissue P-Systems
Parallel Simulation of Probabilistic P Systems on Multicore Platforms (Martínez del Amor et al., 2012)	Martínez del Amor, Miguel Angel et al.	OpenMP, P-lingua, MeCoSim	Probabilistic P Systems especially for modeling Ecosystem
Simulating a P system based efficient solution to SAT by using GPUs (Cecilia et al., 2010a)	Cecilia Jose M. et al.	CUDA	Solution for SAT
SNUPS (Buiu et al., 2011)	Octavian Arsene, Catalin Buiu and Nirvana Popescu	Java	Numerical Membrane Computing
A P–Lingua based Simulator for Spiking Neural P Systems (Macías-Ramos et al., 2011)	Macías–Ramos, Luis F., et al.	P-Lingua	Spiking Neural Networks

JPlant (Rivero-Gil et al., 2011)	Elena Rivero-Gil, et al.	Java	Generating Graphics
A Spiking Neural P system simulator based on CUDA (Cabarle et al., 2011b)	Francis Cabarle, Henry Adorna, and Miguel A. Martinez-del-Amor.	CUDA C Python	Spiking Neural P System
An improved GPU simulator for spiking neural P systems (Cabarle et al., 2011c)	Francis Cabarle, Henry Adorna, and Miguel A. Martinez-del-Amor.	CUDA C Python	Spiking Neural P System
A Java-Based P-Lingua Simulator for Enzymatic Numerical P Systems (ENPS) (Garcıa-Quismondo, 2013)	M García-Quismondo et al.	Java, P-Lingua	Biological Process
DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution (Martínez-del Amor et al., 2012)	M.A. Martínez-del-Amo	CUDA, P-Lingua	Population Dynamics P Systems Environment Ecology
A GPU Simulator for Enzymatic Numerical P Systems (ENPS) models in CUDA (García Quismondo et al., 2012b)	M García-Quismondo et al.	Java, P-Lingua	Biological Process
A GPU Simulation for Evolution-Communication P Systems with Energy Having no Antiport Rules (Bangalan et al., 2013)	Zylynn F Bangalan	CUDA C, P-lingua	Evolution-Communication P Systems
Simulating a Family of Tissue P Systems Solving SAT on the GPU (Martínez del Amor et al., 2013)	M A Martínez del Amor et al.	CUDA	Tissue P System
Accelerated simulation of membrane computing to solve the n-queens problem on multi-core (Maroosi and Muniyandi, 2013)	Maroosi Ali and Ravie Chandren Muniyandi.	Visual C++	N Queens Problem

A C++ Simulator for PGSP Systems (García-Quismondo et al., 2014)	M García-Quismondo et al.	C++	PGSP, Biological Process
A P-lingua based for Tissue P System with cell separation (Perez-Hurtado et al., 2014)	Ignacio Perez-Hurtado, et al	P-Lingua	Tissue P System
Simulating Spiking Neural P systems without delays using GPUs (Cabarle et al., 2011a)	Francis Cabarle, Henry Adorna, and Miguel A. Martinez-del-Amor.	CUDA , Pyhon	Spiking Neural P System
Antibiotic Resistance Evolution Simulator (ARES) (Campos et al., 2015)	Marcelino Campos , et al.	Java, P-Lingua	Biological Processes
P-Lingua Based Simulator for P Systems with Symport/Antiport Rules (Macías-Ramos et al., 2015)	Luis F Macías-Ramos., et al.	P-Lingua	Generic
Lulu - A software simulator for P colonies (Florea and Buiu, 2015, 2016)	Andrei George Florea, Catalin Buiu	Python	P Colonies
Enhancing the Simulation of Membrane System on the GPU for the N-Queens Problem (Ravie and Ali, 2015)	Ravie Chandren and Maroosi Ali.	Visual C++	N Queens Problem
PeP (Florea and Buiu, 2017, 2018)	Andrei George Florea, Catalin Buiu	Python	ENPS

Table 2.1 Tools for specific P Systems

2.5.2 P System tools that are generic in nature

There are a few tools which simulate the P System and allow the users to understand the working of P System. These system are not designed for specific type of P System. This does not mean that all the P System types can be realized using these tools, however it supports more than one or basic simulation of membrane computing. This set of simulators not only includes the simulators used for computational purposes but also includes the

tools that are used for biological application. Thus, these tools are not too specific about the types of P system they can be used for.

Though there are a few in the list, the one which can really be called as generic tool is P-Lingua (Díaz Pernil et al., 2008). This is one of the best framework that allows the users to create any kind of simulator according to their need. This simulator is based on Java. Using P-Lingua there has been a tool developed, which is called MeCoSim (Pérez-Hurtado et al., 2010). This tool gives a user, a wide range option and allows the users to solve and simulate several kinds of computational problems; it also allows the users to use it for biological processes. Based on P-Lingua, several simulators have been designed. According to RGNC, there is PMGGPU project by Research Group in Natural Computing, University of Seville. This project uses primarily P-Lingua to design simulators using Graphics Processing Unit (GPU); primarily designed with CUDA, these simulators use GPU for realizing different kinds of P Systems. The generic tools are listed in table 2.5.2.

Tool/ Software	**Name of the developers**	**Base tool /Framework /Language**	**Purpose/ Application**
Web-PS: Web based simulator for Membrane Computing (Bonchiş et al., 2005)	Cosmin Bonchi, Cornel Izba, Dana Petcu, Gabriel Ciobanu	Embedded C, CLIPS	Web Based Simulator
SL_P Simulator (Gheorghe, 2010)	M Gheorghe et. al	Scilab	Biological Processes
C_PSimulator (Gheorghe, 2010)	M Gheorghe et. al	C	Biological Processes
PSim (Bianco et al., 2007)	Luca Bianco et al.	Java	Bio-Systems
Cyto-Sim: Biological compartment simulator (Sedwards and Mazza, 2007)	S Sedwards et al.	J#	Bio-Systems

P-Lingua 4.0: a programming language for Membrane (Díaz Pernil et al., 2008)	Daniel Díaz Pernil et al.	P-Lingua Core	Generic P System
MeCoSim: Membrame Computing Simulator (Pérez-Hurtado et al., 2010)	Ignacio Perez-Hurtado, et al.	P-lingua	Generic
Infobiotics Workbench (Blakes et al., 2011)	Jonathan Blakes et al.	Jmcss-SBML, Standalone software	Generic Tool for Biological aspects of membrane computing
Improved implementation of simulation for membrane computing on the graphic processing unit (Maroosi et al., 2013)	Maroosi Ali et al.	CUDA, C++	General
MeCoGUI: A Simple, Java-Based Graphic User Interface for P-Lingua (Garcia-Quismondo)	M García-Quismondo et al.	Java, P-Lingua	Generic

Table 2.2 Generic tools for P Systems

2.5.3 Analysis

This section quantitatively analyses the tools. Though the number of tools developed for computational aspect of Membrane computing is more, there have been considerable development for biological processes off late. Though the number of tools that have been developed for biological processes are less, there has been a considerable amount of research going on in this area.

There is an increase in computational simulation in the later years because of the development in P-lingua. As a language / framework, one of the most used language / framework is P-lingua. After its development there has been consistent work in this area and

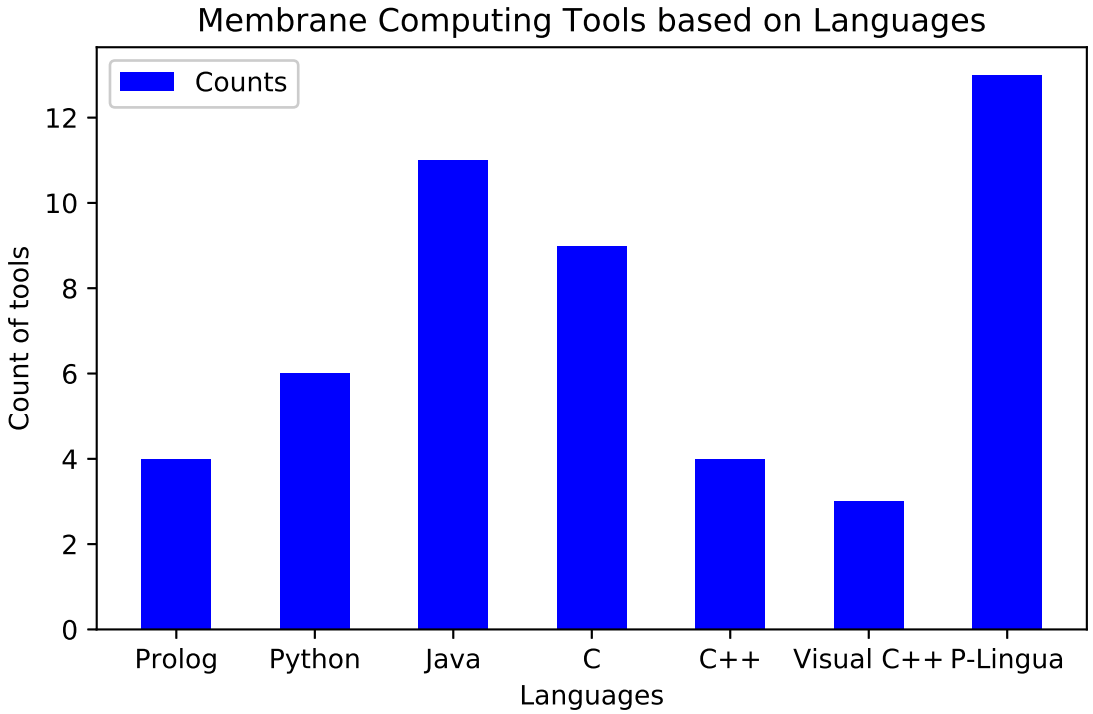

Figure 2.3 Tools based on their language/framework

several tools have been created using this framework (Figure 2.3). Though this framework is developed by using Java, P-lingua based tool has not been included for classification under Java so as to know specifically the number of tools that exclusively use P-lingua and the tools which exclusive use only Java (Not Part of P-lingua).

From the tables in the previous sections (Table 2.5.1 and Table 2.5.2) it is seen that, there is a rise in P-lingua in recent years, such that no other language or framework is preferred, except C++ for sequential simulation and Python (Florea and Buiu, 2015). In many places CUDA with C is used but it is mainly being used with P-lingua.

From the review of the tools, related to membranes, it is perceived that for ENPS there are only few tools available. There has been significant works, though less in number, that deal with the simulation of NPS (specifically ENPS). Pavel et al. (2010) have introduced the ENPS, the membrane computing model for which this paper also proposes a Java based Simulator called SNUPS (Pavel et al., 2010; Buiu et al., 2011), a software implementation of ENPS. It is the first tool developed for ENPS. It is a desktop application with two major

components: A GUI and a computation model. SNUPS, as a batch application, takes the input of the membrane model as an XML file and gives output in GUI. The computation engine of the simulator uses multi-threading in Java.

Another tool named Sim P has been developed by Brandusa Pavel (2011). This is again a Java-based tool for ENPS execution. This tool has first been used for robot localization problem. This tool is a significant contribution, for two important reasons: This is one of the first tools that is used for a practical real world problem (i.e. Robot localization (Pavel and Buiu, 2012; Pavel et al., 2012)) and the core structure of SimP is used by another Java-based tool for ENPS (Garcıa-Quismondo, 2013). Following this, a Java-based tool has been developed (Garcıa-Quismondo, 2013). This tool is one of the best Java-based implementations of ENPS. The tool requires the input (membrane structure) to be given in XML format.

After this is the era where an attempt has been made to exploit parallelism in the model. Quismondo et al. (García Quismondo et al., 2012a; García-Quismondo et al., 2013) have proposed a GPU based simulator for ENPS. The simulator is written in CUDA C. This simulator aims to provide a significant acceleration of execution times, as compared to the then state-of-the-art sequential ENPS simulators. The paper provides a detailed account of the structural and functional design of the simulator. This is an important development in the area of simulation and modelling for ENPS, as it uses GPU for the first time. This is significant work, as membrane computing is a parallel paradigm (structure) and simulating this in an environment that supports parallelism is the most appropriate thing to do. This tool supports XML based input for membrane structure. With this tool, the authors have also developed a sequential counterpart to facilitate the tool users to analyze a designed membrane in both sequential and parallel manner. Further, in another work, the authors elaborate on the use of the GPU based tool for large scale problems (García Quismondo et al., 2012a). There is also work that use GPU's for simulation (Cabarle et al., 2019; Maroosi et al., 2014), which are for different variants of P Systems (not ENPS) and different applications. Further there is a work that elaborates on the existing simulators for membrane computing, which use GPU's for simulation (Martínez-del Amor et al., 2015).

There is, in recent times, significant work in the area of ENPS implementation by Pérez-Hurtado et al. The authors propose ENPS based model for Rapidly Exploring Random Tree (RRT) and its variant RRT*, which are used for robotic path planning. The proposed solutions are emulated (using parallelism). There are two implementations based on CUDA and OPENMP, where a speedup upto 24x is achieved, against the best multi-threading configuration and a speedup of upto 6x is achieved against sequential implementation, respectively (Pérez-Hurtado et al.).

Until now, all the tools discussed till now for ENPS are either based on C/C++ or Java and using of them use XML based structure for representing membranes. In 2017, Florea and Buiu (2017, 2018) have developed a Python-based open source simulator for ENPS called PeP Simulator. There are certain features of the tool that makes it different from others. First being its development in Python. Second is its ease of use (no installation other than Python is required). The third is an easy way of giving input i.e. the input is given as a pep file, which is a membrane representation in a human-readable format. This tool is easy to use, but has a drawback that it supports only sequential execution and doesn't support real-time parallel execution.

This study has lead to the development of a simulator to maintain the ease of use of PeP simulator with efficient parallel execution, by preserving the actual parallel paradigm of ENPS. It uses the same input format and interface as defined by Florea et al. (Florea and Buiu, 2017, 2018) to maintain compatibility between the serial system and the developed parallel system. GPU's are used with CUDA in C, to enable parallelism and to achieve greater speed up. This work has been elaborated in the upcoming chapters.

2.6 Workflow Scheduling Algorithm in Cloud

Workflows are an integral part of cloud ecosystem. Out of several important processes associated with workflows, scheduling tends to be an important one. A proper scheduling approach is an important component of a good workflow management system, as it ensures optimal utilization of resources. Many researchers have proposed several heuristic and meta-heuristic algorithms for scheduling cloud workflows. Often the objectives considered

are makespan, cost or energy.

There are several techniques (Heuristics and Meta-Heuristics) that have been used. Heuristics is a class of techniques where, according to the nature of the problem, several specific techniques are applied to solve the problem more efficiently and quickly (Wu et al., 2014; Rodriguez and Buyya, 2014; Arabnejad et al., 2018). Usually, these techniques are used for problems where exhaustive solutions are a time consuming process (Many real world problems). The other class of solutions, called meta-heuristic approaches which is similar to heuristic for having optimal solutions that aims to solve the problem in an non-exhaustive and often in a probabilistic manner. There is a primary difference between heuristics and meta-heuristics, the former being problem specific and the latter not being so. Meta-heuristic approaches are primarily used where there is no exhaustive solution possible and there is a need of approach that can approximately solve the problem with the most optimal value. The problem of workflow scheduling considered in this study, is a similar problem where obtaining exhaustive solutions is not possible. There are several meta-heuristic approaches for solving this, out of which many are bio-inspired approaches.

The final objective of the work is to propose a workflow scheduler, based on membrane computing paradigm. Workflow scheduling is a NP−Complete problem. There are many heuristic and meta-heuristic algorithms that have been proposed for this problem in literature (Pandey et al., 2010; Wu et al., 2014; Zhu et al., 2015; Kumar and Sharma, 2018). Usually, heuristics find out solution faster than meta-heuristic where as the latter consumes more time due to exploration of solution space. The advantage of meta-heuristic, over heuristic is its generality, thereby, allowing us to apply the algorithm to be applied anywhere, wherever it is suitable. There have been many works in this area and these works have been discussed as follows:

Calheiros and Buyya (2013) have proposed Enhanced IC-PCP with Replication (EIPR) algorithm which completes user tasks within given deadline in public cloud. Deadline constraint in smaller budget is the primary concern of this algorithm. They divided this problem into two sub-problems: of provisioning and scheduling where provisioning deals with finding optimal number of virtual machines whereas scheduling problem deals with map-

ping the order of allocation of tasks to virtual machines. The authors have used Cloudsim as a simulator. EIPR gives better results as compared to IC-PCP algorithm.

Zhang et al. (2014a) have proposed a new method, Iterative Ordinal Optimization (IOO) method. The proposed method is used to get best overhead analysis such as time and space complexity in workflow scheduling by using iterative approach. IOO method proves to be better than the algorithms compared. Another work related to workflow scheduling is presented by Rodriguez and Buyya (2014) using a meta-heuristic algorithm, Particle Swarm Optimization (PSO) is used for resource provisioning and task scheduling on scientific workflows on IaaS, thereby minimizing execution time and meeting deadline constraint. They have used cloudsim as a simulator. This algorithm has been compared with several other algorithms, namely, IC-PCP, PSO, and SCS and has proved to be better than them.

Another work related to workflow scheduling is presented by Wu et al. (2014) who have proposed a heuristic algorithm named Critical Greedy Algorithm. This algorithms main goal is to reduce the workflow's end to end delay under financial constraint. This has been implemented in cloudsim. Zhu et al. (2015) have proposed an Evolutionary Multi-objective workflow scheduling algorithm which optimizes both cost and makespan on IaaS. This algorithm gives better results when compared with SPEA2, MODE, NSPSO and Modified Heterogeneous Earliest Finish Time (MOHEFT).

Li et al. (2015) have developed a Cost and Energy Aware Scheduler (CEAS) algorithm to minimize the execution cost and reduce energy consumption, by meeting deadline constraint. CEAS is a sequence of four algorithms which are VM selection algorithm, sequence tasks merging algorithm, parallel tasks merging algorithm and VM reuse algorithm. The experiment is conducted by using cloudsim. This final algorithm is compared with HEFT, MOHEFT, EES and EHEFT algorithms for energy efficiency and proves to be better than them.

Rimal and Maier (2016) have proposed a novel approach for cloud based workflow scheduling in compute intensive workflow applications. The main objective of the algorithm is to minimize the makespan, cost of execution of workflows and utilize idle re-

sources. The proposed algorithm is compared with First Come First Serve (FCFS), Easy Backfilling and Minimum Completion Time (MCT) algorithms to evaluate performances. Simulation has been done on Cloudsim simulator and the proposed algorithm proves to be better than others. Chen et al. (2017a) have proposed a novel scheduling approach Selective Tasks Duplication (SOLID). SOLID−R, Earliest Finish Time - Maximum Effective Reduction (EFT−MER) and EFT-MER with Exploring Laxity Time (EFT−MER−EL) algorithms are compared for the performance evaluation of the proposed approach. Yao et al. (2017) proposed a novel algorithm ICFWS (Fault Tolerant Workflow Scheduling). This algorithm is evaluated on workflowsim toolkit. ICFWS is compared with Task Completion Rate (TCR) and VMs Reserve Time Rate (VRTR) and both the algorithms.

Wu et al. (2017) have proposed a meta-heuristic algorithm L-Ant Colony Optimization (L-ACO). The main objective of the algorithm is to minimize execution cost of workflow by meeting deadline constraint. L-ACO algorithm gives better performance when compared to the IC-PCP, PSO and PROLIS. Chen et al. (2017a) worked on a method for workflow scheduling called as Multi-Objective Ant Colony System (MOACS) approach for cloud workflow scheduling. In this paper MOACS approach is compared with HEFT, Modified HEFT, EMS-C, NSGA - II. MOACS gives best performance when it compares to these five algorithms. Arabnejad et al. (2018) proposed a heuristic algorithm which is budget and deadline aware and works for IaaS clouds.

Verma and Kaushal (2017) have proposed a non-dominance sort based Hybrid Particle Swarm optimization (HPSO). The main objectives of this algorithm are makespan and cost under deadline and budget constraints. For this study cloudsim has been used. This algorithm performs better than MOPSO, FDPSO, NSGA -II algorithms. Pandey et al. (2010) have proposed a PSO based Heuristic Algorithm. This algorithm shows that it is three times better than Best Resource Selection (BRS). The main parameter of this algorithm is only Execution Time. This is one of the significant works in the area of workflow scheduling in cloud.

Wu et al. (2010) have proposed Revised Discrete Particle Swarm Optimization (RDPSO) Algorithm which schedules workflow applications that considers data computation costs

and transfer costs. This algorithm performs better than standard PSO and BRS algorithms. Tasks are executed on amazon compute cloud. Adhikari and Amgoth (2019) have proposed intelligent water drop based workflow scheduling in IaaS cloud. The main objective of this algorithm is to minimize the execution time of a workflow and improve utilization of VM with the given budget and deadline constraint. This algorithm is better than SCS, IC-PCP, Intelligent Water Drop (IWD), PTS and Intelligent Water Drop based Dynamic Workflow Scheduling (IWD-DWS).

Liang et al. (2014) have proposed a meta-heuristic algorithm Artificial Bee Colony for Workflow Scheduling. The main objective of this algorithm is to minimize the makespan and the implementation succeeds in doing so. Casas et al. (2018) proposed GA-ETI algorithm. The main objective of this algorithm is to reduce the makespan. GA-ETI gives better performance when compared to HEFT, provenance, FSV (Flexible Selection of VMs) algorithms.

Nasonov et al. (2014) have proposed a hybrid algorithm which combines both HEFT and genetic algorithm. The main objective is to improve the makespan of workflow, which the algorithm achieves. Mansouri et al. (2019) have proposed hybrid task scheduling strategy Fuzzy System and Modified PSO (FMPSO). The results have been evaluated in cloudsim toolkit. The main objective was to improve makespan, improvement ratio, imbalance degree, efficiency, and total execution time when compared to other approaches.

Elaziz et al. (2019) have proposed a task scheduling algorithm named hybrid moth search and differential evaluation. The main objective of this approach is to improve the makespan. This MSDE algorithm has been simulated in cloudsim environment. There is also work by Mao and Humphrey (2011), who have proposed an auto scaling approach. The main objective is to reduce the cost and meet the deadlines.

Kumar and Sharma (2018) have proposed a PSO-COGENT algorithm. The main objective is not only to optimize the execution cost and time but also to reduce energy consumption by considering deadline as a constraint and improve the throughput when compared to Honeybee, PSO and Min-Min algorithms. Maciej (2012) have proposed static and dynamic strategies for both task scheduling and resource provisioning. This paper

discusses efficient management of ensembles under budget and deadline parameters.

Adhikari and Amgoth (2016) have proposed an algorithm named Efficient Workflow Scheduling Algorithm (EWSA). The objective of this algorithm is to dynamically estimate the execution time of all tasks. This algorithm creates minimum number of virtual machines to complete the entire execution within deadline. Su et al. (2013) have proposed cost efficient task scheduling algorithm using two heuristic strategies. The main objective is to reduce the makespan. This algorithm performs better than SLR, MCR algorithms. Chen et al. (2017b) have proposed an efficient algorithm named Minimizing Schedule Length using Budget Level (MSLBL). The main objective of this algorithm is minimizing the schedule length of application by satisfying budget constraint. It gives better performance when compared to the state of art HBCS algorithm.

Abrishami et al. (2013) have proposed a PCP algorithm for cloud environment, with two algorithms for scheduling. In the first phase, the algorithm which is called IC-PCP and in the second phase the algorithm is called IC-PCPD2. IC-PCP outperforms IC-LOSS in all cases. However, IC-PCPD2 is better than IC-LOSS in Montage and Epigenomics workflow. Abrishami and Naghibzadeh (2012) proposed QoS based workflow scheduling PCP. The main objective of this algorithm is to minimize the cost of workflow execution, while meeting budget constraint.

Byun et al. (2011) have proposed Partitioned Balanced Time Scheduling (PBTS) Algorithm. The main objective of this algorithm is to minimize the financial cost and maximize the utilization component of the resources. In this paper PBTS is compared with theoretical optimal solver and Balanced Time Scheduling (BTS). Casas et al. (2017) have proposed a Balanced and file Reuse-Replication Scheduling (BaaRS) algorithm. The main objective of this algorithm is the execution time and monetary cost of running scientific workflow. This algorithm is implemented in VMware-ESXi-based (Version 5.5) private cloud and it outperforms provenance algorithm.

There is an important method for the heterogeneous machines that is one of the first heuristic method for workflow scheduling (Topcuoglu et al., 2002). The parameter considered for optimization is makespan of the workflow. The heuristic method proposed is one

of the best heuristics for workflow scheduling over heterogeneous components and it has given better results than other standard methods of that time and can be considered as one of the best methods till date. This methods logic is in the present study is considered to develop an approach that uses strict membrane computing structure for workflow scheduling in cloud.

Apart of these works there is an important work which has come in late 2014, where, for the first time Membrane computing model was used for Workflow Scheduling in Cloud (Ahmed et al., 2014). The developed model though based on membrane computing is quite different from the model that is proposed in this thesis. Ahmed et al. (2014) proposed a membrane inspired model that imbibes the structure of P System but it doesn't follow all the membrane-based execution rules. It is a workflow scheduling approach uses only the hierarchical structure of membrane instead of using P System as a computing paradigm by following it strict rules.

It can be inferred from the literature review that there have been many works which use heuristic approaches for solving the problem of workflow scheduling. Further analysis on the works reveal that among several categories of algorithm the Bio-inspired approaches tend to be effective in dealing with workflow based scheduling algorithms, if used appropriately. Getting a cue from the above literature and after a detailed analysis of several heuristic and meta-heuristic approaches, primarily bio-inspired approaches; the present work proceed towards selecting membrane-based algorithm to be used for workflow scheduling in cloud. Further the behaviour of the used algorithm and its suitability for the proposed problem is analysed before proceeding with implementation in chapter 6.

2.7 Summary

An extensive study related to four important components of the work; Service Selection in cloud, Membrane computing applications and Numerical P System, Tools for Membrane Computing and Workflow Scheduling methods in cloud, is done. In addition to the works in the area, this section elaborates on the background information that is required to proceed further. All these four components are important and the study of each com-

ponent leads to specific research gap which is filled by using our contribution. Thus, the two wide fields of cloud and membrane computing are related and after a detailed study of these components the problem statement is defined which gives one of the best solution for cloud service selection and workflow scheduling through membrane computing paradigm. Further, new tools are developed, which fill the gaps that have been identified based on survey of tools available for membrane computing.

Chapter 3

P SYSTEM BASED SERVICE SELECTION MECHANISM

Service selection is a process of selecting the best service (or items), given a set of services available. The selection process involves ranking of services. This chapter elaborates on two important service selection approaches based on amalgamation of Multi-criteria Decision Making (MCDM) and Enzymatic Numerical P System (ENPS). The two approaches that have been proposed are ENPS-IAHP and ENPS-IPROMETHEE. Both the approaches are suitable for service selection, but each has a specific purpose. The first approach is designed specifically to calculate weight, given the user preferences. The second method is used for service selection in general (services considered as items/alternatives). The proposed method is analysed and the obtained results show IPROMETHEE to be sensitive in nature.

3.1 Introduction

Service selection is the process of ranking a given set of services based on attribute (feature) values. These attributes are the features with certain weights assigned for each value. In general, the most common method used for solving this kind of problem is called Multi-Criteria Decision Making (MCDM). The services in this particular case are not necessarily cloud based services, but are some collection of items/alternatives, which have attributes associated with it, and which can be selected quantitatively.

In the study, MCDM structure is considered as a base for the model being proposed. The aim is to propose a parallel model for solving service selection problem (ranking the services). An inherently parallel model that can be combined with MCDM properties, to obtain a new, wholly parallel approach specific to this problem is chosen. MCDM model is considered as the base solution for its structure and proved competence in the area of cloud service selection. Based on the structure of service selection problem (MCDM), membrane computing based models are found to be suitable, to solve this problem with properties like; simple inherent parallel structure, Turing Universality, multiple variants, variety of applications and its developing software support.

Gheorghe Paun has introduced membrane computing in his seminal paper (Păun, 2000). The devices used to realize Membrane computing are called as P Systems. Membranes are inspired by nature; in particular, by a living cell. A membrane model (cell like) has a hierarchical structure of membranes, where the internal components are disjoint, as in figure 2.2. The outer layer is called as skin membrane and all the other membranes are contained in it. There can be any number of membranes present inside the skin and all these membranes can communicate with each other, i.e. they can pass information among themselves. This structure inherently supports parallelism, based on which the whole design works. Unlike other parallel models that have a limited scope or constrained scope of application, P System has a wider range of applications due to it being a computational paradigm. Further description is given in the latter part of the work. There are numerous variants of P Systems available (Paun et al., 2010). Each variant has a different structure and is designed for a different purpose. One such specific variant called Enzymatic Numerical P System (ENPS) is used for the chosen problem of Multi-criteria Decision Making (for cloud service selection); the reason for selection of this ENPS and its structure is discussed in literature, section 2.1.

3.2 Enzymatic Numerical P System based Improved Analytical Hierarchy Process (ENPS-IAHP)

As part of this the concept of parallelizing an enhancement of Analytic Hierarchy Process (AHP) is proposed. AHP has first been developed by L Saaty et al. in 1982 (Saaty, 1983, 1988). It is used in the domain of decision making. It is a simple method which facilitates the user to decide about the selection among the choices (items) available. Each and every choice consists of several attributes $1-n$. Each and every attribute has weight to be given according to the choices (items) to be selected. AHP allows the user to calculate the weights of the attributes by knowing the dominance of each attribute over the other. Thus, by using these dominance values the AHP method formalizes the weights, checks for its consistency and proceeds with calculation of ranks of the choices. This has very wide applications as mentioned by Vaidya and Kumar (2006). The improved version of AHP called as Improved Analytic Hierarchy Process (Rao, 2007) is used here and its steps are elaborated.

3.2.1 Improved Analytic Hierarchy Process (IAHP)

IAHP model, as the name suggests, is an improvement over the existing AHP model. The first step is to identify the attributes that are to be used for the considered problem. After all the values associated with attributes are finalized, based on any data or by the users experience, the user may proceed with the method.

This problem is divided into three sub-problems which are described as follows:

Sub-problem 1: Calculating the weights using the comparison matrix

The comparison matrix (Preference matrix) is a matrix in which each attribute is rated against each other attribute for its importance. The size of the matrix is equal to *nxn*, where *n* is the number of attributes that are considered. There are nine allowed values that can be entered in the matrix, *m* are the attributes in the row and *n* are the attributes in the column. This matrix is labelled as A_1. Further, the reciprocal of the values is entered by reversing the position of rows (position i) and column (position j) of the matrix.

1. If an attribute in position i (row) is extremely important than the attribute j (column), then the value entered is 9 in (i,j) position.

2. If an attribute in position i (row) is highly important than the attribute j (column), then the value entered is 7 in (i,j) position.

3. If an attribute in position i (row) is important than the attribute j (column), then the value entered is 5 in (i,j) position.

4. If an attribute in position i (row) is somewhat important than the attribute j (column), then the value entered is 3 in (i,j) position.

5. If an attribute in position i (row) has same preference as of attribute j (column), then the value entered is 1 in (i,j) position.

$$A_{1_{MXM}} = \begin{pmatrix} r^{11} & r^{12} & \dots & r^{1M} \\ r^{21} & r^{22} & \dots & r^{2M} \\ r^{31} & r^{32} & \dots & r^{3M} \\ \dots & \dots & \ddots & \dots \\ r^{M1} & r^{M2} & \dots & r^{MM} \end{pmatrix}$$

Find the relative normalized weight of each attribute by calculating the geometric mean of the i^{th} row.

$$GM_j = [\prod_{j=1}^{l} K_{ij}]^{1/M} \tag{3.1}$$

In the above sub-problem all the values of a single row are multiplied and are raised to the power of (1/M) where M is the number of columns (attributes). Further, the geometric

means of rows in the comparison matrix are normalized.

$$w_j = GM_j / \sum_{j=1}^{l} GM_j \tag{3.2}$$

The geometric mean method of AHP is used because it is easy to use. w_j represents the final weight.

Sub-Problem 2: Consistency of weights

After obtaining the weights the next major operation is to check the consistency of the obtained weight. There are several steps to be followed:

Calculate A_2 vector. A_2 is assigned as vector with value $[w_1, w_2 \cdots, w_n]^T$ and further $A3$ matrix is calculated by multiplying A_1 and A_2.
As a next step, A_4 is calculated, which is equal to A_3/A_2.
Followed by that Eigen Value (λ_{max}) is calculated.
λ_{max} = Average of Matrix A4 (Rao, 2007).
The penultimate step involves calculating CI, Index CI = (λ_{max} - M)/ (M-1).
The smaller the value of CI, the smaller is the deviation from consistency.
Select the Random Index (RI) value from the number of attributes used in the decision making. The RI values are given in the table 3.1 (Rao, 2007).

Finally, Consistency Ratio (CR) is calculated value: CR = CI/RI. If the CR value is less than 0.1 then the values that are obtained as weight are correct, otherwise the values need to be re-checked. The consistency in CR value indirectly denotes that the relative importance matrix is proper. This Improved AHP method is characterized by numerical calculation which predominantly uses real values. One more characteristics of this method is that, this method involves several rules based on the number of items that have been selected. This methods though cannot be considered completely independent as each and every sub-problem is dependent on each other. However, each sub-problem within itself can be efficiently parallelised. As the number of items increases, for each sub-problem the parallelization proves to be more effective.

Attributes	1	2	3	4	5	6	7	8	9	10
RI	0	0	0.52	0.89	1.11	1.25	1.35	1.4	1.45	1.49

Table 3.1 Standard Random Index (RI) values

3.2.2 ENPS-IAHP Membrane Structure

IAHP can be solved using numerical P Systems, specifically ENPS. The IAHP is divided into three sub-problems and these three sub-problems are inter related. Each and every consecutive sub-problem is dependent on the previous sub-problem as the final result of the former is used by the current sub-problem. The sub-problems are:

- Computing Weights
- Verifying the consistency of weights
- Obtaining Ranks of the items

The process is started by computing weights in IAHP, using the given preference matrix. Out of the above sub-problems, the sub-problem 2 is directly dependent on the first sub-problem. The second sub-problem checks the consistency of the value obtained during the first sub-problem. Only if the values obtained are consistent according the rules then it is advisable to proceed further, for it is not advisable to proceed further as mathematically there can be a lot of deviation in the values chosen by the user. A high deviation means that the whole assignment process is wrong and the user has assigned the values that have been inconsistent; the user has to change the preference matrix, so that preference matrix is logically consistent. Once the user has verified that the values are consistent then the user can proceed with the sub-problem 3. So, if the user is inconsistent in giving the preference matrix and it is found that in sub-problem 2 then the process of sub-problem 1 has to be computed again. The solution for sub-problems for ENPS-IAHP are as follows:

Membrane System for Sub-problem - 1

The sub-problem 1 membrane system consists of two membranes, the membrane 1 and membrane 2 (as in figure 3.1). The membrane (M_1) and (M_2) consists of several rules. As

per the conditions of ENP System, each membrane can have more than one rule. All the programs can be controlled by the enzymes. The presence of enzyme allows more than one rule to be executed simultaneously based on the values of the enzyme. These enzyme control the execution and flow of the program. Hence using these enzymes, it is possible to control the selectively execute the rules (programs).

The membrane 1 (as in figure 3.1) reprsents the first step of IAHP. The programs are numbered as $Pr_i, where \ \ i \geq 1$. The primary way to determine the size of the Improved AHP problem is by the number of attributes that have been compared. Consider number of attributes that are being compared in IAHP as n. The number of programs in this are dependent on the value of n. The figure 3.1 shows membrane system.

The first membrane calculates the geometric mean of each row of preference matrix that is available. The variables $x_{1,1}, \ldots, z_{n,1}$ are assigned values of the preference matrix in row-wise manner. This convention has been followed through out this membrane as well as other membrane systems. Accordingly, the resultant values are stored in set of variables. The next membrane receives calculated values from membrane one and then does normalization of the received values followed by calculation the weights.

Role of Enzymes:

The enzyme plays an important role in this membrane system. The enzyme e_1 is used to accommodate more than one active productions, whereas enzyme e_2 is used to control the flow of the program. The second membrane uses e_2, therefore the programs in the second membrane gets executed only after cycle step 1. In the first step of the cycle all the programs related to the first membrane are executed, as 3 out of 4 programs are controlled by enzyme e_1 and the initial value of this enzyme is assigned to l (Any value which is always strictly greater than the smallest variable must work). The fourth program available in the membrane is not controlled by any enzyme and it always remains active. Thus, this enzyme is executed devoid of any restrictions.

The second membrane consists of three programs which are controlled by enzyme e_2, and the initial value of the enzyme is assigned to 0. Thus these programs do not execute

in the first step of the cycle. This has been done as the values of the variables that have been used in the production functions are dependent on the repartition protocols of the first membrane and this execution of membrane 2 is meaningless until and unless membrane 1 has been executed. Thus, program $Pr_{4,1}$ controls the execution of the second membrane and only when this active membrane is executed, atleast once, the other membrane programs can get executed.

$\underline{Sub-problem\,1\,-\,M_1}$

$$x_{1,1}[value], \ldots, x_{n,1}[value]$$
$$\cdots$$
$$\cdots$$
$$z_{1,1}[value], \ldots, z_{n,1}[value], e_1[l]$$
$$Pr_1 :\ 2 \times [(x_{1,1} \times \cdots \times x_{n,1}]^{1/n}\ \langle e_1 \rightarrow \rangle\ \ 1 \mid x_{1,2}+1 \mid x_{n+1,2}$$
$$\cdots$$
$$\cdots$$
$$Pr_n :\ 2 \times [(z_{1,1} \times \cdots \times z_{n,1})]^{1/n}\ \langle e_1 \rightarrow \rangle\ \ 1 \mid x_{n,2}+1 \mid x_{n+1,2}$$
$$l\ \rightarrow\ 1 \mid e_2$$

$\underline{Sub-problem\,1\,-\,M_2}$

$$x_{1,2}[0], \cdots, x_{2n+1,2}[0], e_2[0]$$
$$Pr_{1,1} :\ x_{1,2} \div x_{n+1,2}\ \langle e_2 \rightarrow \rangle\ \ 1 \mid x_{n+2,2}$$
$$\cdots$$
$$\cdots$$
$$Pr_{n,1} :\ x_{n,2} \div x_{n+1,2}\ \langle e_2 \rightarrow \rangle\ \ 1 \mid x_{2n+1,2}$$

Figure 3.1 Membrane System for Sub-problem 1

Membrane System for Sub-problem - 2

The Sub-problem 2 consist of two membranes (as in figure 3.2). The purpose of this membrane system is to check whether the weights, that have been obtained after the first sub-problem are consistent or not. On the whole, this membrane system consists of $4n+4$ programs out of which $4n+1$ programs are available in the membrane M_1 and other programs are available in membrane M_2. There are total five enzymes which control the flow of execution, enzyme e_1 and e_2 are used in M_1 while e_3, e_4 and e_5 are used in M_2. The variables $x_{1,1}, \ldots, z_{n,1}$ contain the values of the preference matrix in row-wise manner and

accordingly, the required variables for operation have been used and assigned.

The first n membranes are used to calculate the matrix A_3 which is obtained my multiplying the comparison matrix and the transpose of vector consisting of the weight calculated in the sub-problem 1. Further, $2n$ programs are used for retaining the value of weights and the results obtained in the previous calculation. Mathematically the process involves dividing the result by matrix A_2, this process is done by using the next n programs. The membrane M_2 receives the required values (Vector A_4) from membrane M_1 and further computes average of the Eigen vector. The next two programs in sequential steps calculate the Consistency Ratio (CR), which must be less than 0.1 for the sub-problem 1 to be correct.

Role of Enzymes:

There are five enzymes being used. Each enzyme is assigned to a certain set of programs for controlling its execution. The enzyme e_1 is assigned to a total of $3n$ programs and these $3n$ programs are always executed as enzymes remain active from the step 1 of the cycle. The enzyme e_2 is assigned to n programs which always execute after step 1 in the cycle. Similarly e_3, e_4, and e_5 are assigned to a program each and execute as step 3, step 4 and step 5 respectively.

Finally the Consistency Ratio (CR) is obtained, which determines whether the process needs to be proceeded further. If ($CR < 0.1$) then the comparison matrix, assigned by the user, is proper and control can proceed to the third sub-problem otherwise it is not advisable to proceed without changing the comparison matrix and re-doing the sub-problem 1 and sub-problem 2.

$\underline{Sub-problem2-M_1}$

$$x_{1,1}[value], \ldots, x_{n,1}[value]$$
$$\cdots$$
$$\cdots$$
$$z_{1,1}[value], \ldots, z_{n,1}[value]$$
$$k_{1,1}[value], \ldots, k_{n,1}[value]$$
$$w_{1,1}[value], \ldots, w_{n,1}[value]$$
$$e_1[l], e_2[0]$$

$$Pr_{1,1}: x_{1,1} \times w_{1,1} + \ldots + x_{n,1} \times w_{n,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid k_{1,1}$$
$$\cdots$$
$$\cdots$$
$$Pr_{n,1}: z_{1,1} \times w_{1,1} + \ldots + z_{n,1} \times w_{n,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid k_{n,1}$$

$$Pr_{n+1,1}: k_{1,1} \div w_{1,1} \ \langle e_2 \rightarrow \rangle \ 1 \mid x_{1,2}$$
$$\cdots$$
$$\cdots$$
$$Pr_{2n,1}: k_{n,1} \div w_{n,1} \ \langle e_2 \rightarrow \rangle \ 1 \mid x_{n,2}$$
$$Pr_{2n+1,1}: k_{1,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid k_{1,1}$$
$$\cdots$$
$$Pr_{3n,1}: k_{n,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid k_{n,1}$$

$$Pr_{3n+1,1}: w_{1,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid w_{1,1}$$
$$\cdots$$
$$Pr_{4n,1}: w_{n,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid w_{n,1}$$
$$Pr_{4n+1,1}: 4 \times l \ \rightarrow \ 1 \mid e_2 + 1 \mid e_3 + 1 \mid e_4 + 1 \mid e_5$$

$\underline{Sub-problem2-M_2}$

$$x_{1,2}[0], \cdots, x_{n+3,2}[0], \ x_{n+4,2}[value],$$
$$e_3[-l], \ e_4[-2l], \ e_5[-3l]$$
$$Pr_{1,2}: (x_{1,2} + \cdots + x_{n,2}) \div n \ \langle e_3 \rightarrow \rangle \ 1 \mid x_{n+1,2}$$
$$Pr_{2,2}: (x_{n+1,2} - m) \div (m-1)) \ \langle e_4 \rightarrow \rangle \ 1 \mid x_{n+2,2}$$
$$Pr_{3,2}: x_{n+2,2} \div x_{n+4,2} \ \langle e_5 \rightarrow \rangle \ 1 \mid x_{n+3,2}$$

Figure 3.2 Membrane System for Sub-problem 2

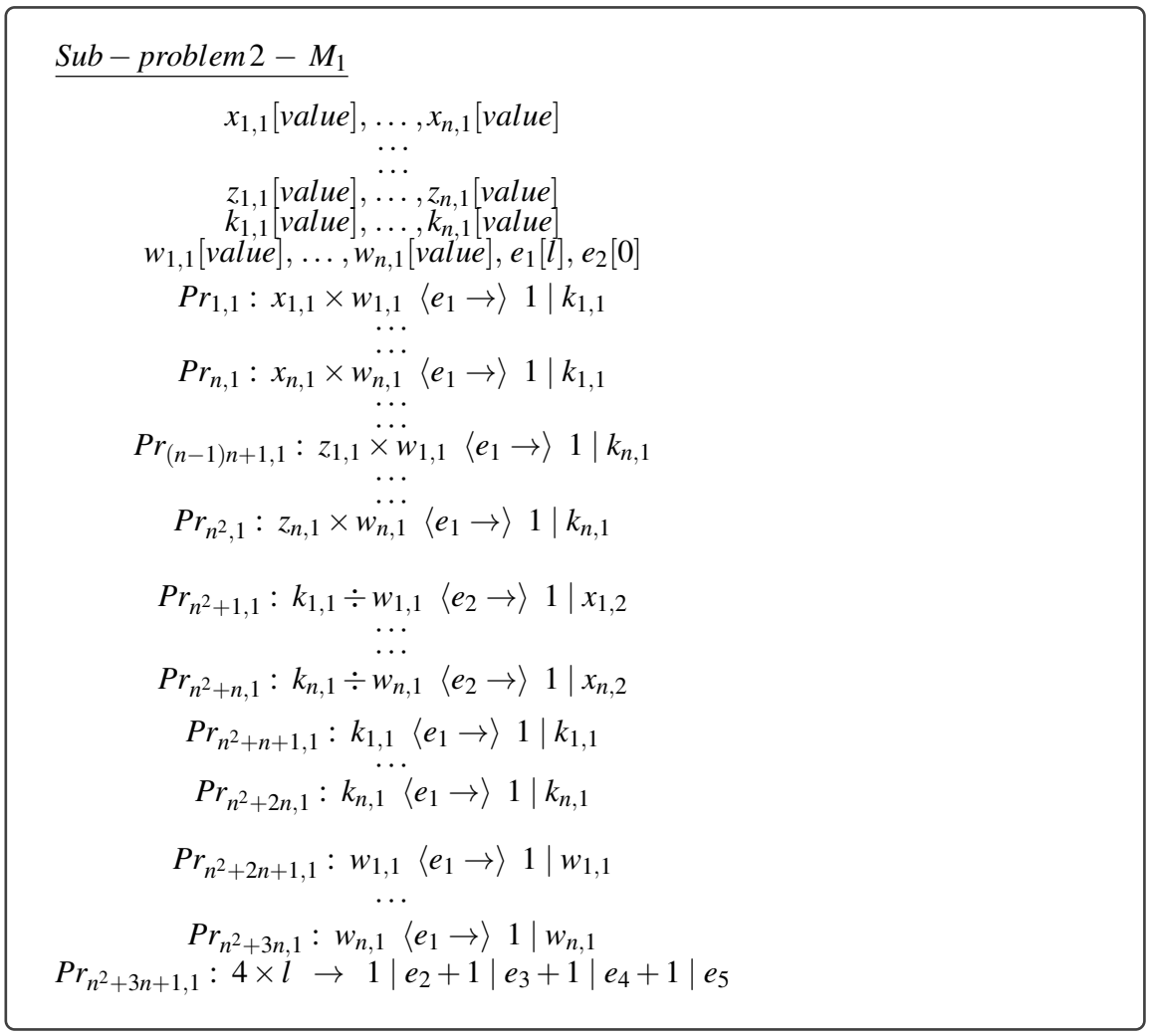

Figure 3.3 Membrane System for Sub-problem 2 - Parallelized Further

Further Parallelization of Sub-problem 2 Membrane 1

The above mentioned method consists comparatively less number of rules. This sub-problem can be parallelized further, where the program $Pr_{1,1}$ *to* $Pr_{n,1}$ can be divided, as in figure 3.3. By doing this, the number of program increase by a factor n, for each existing program. Thus in this case maximal parallelism is achieved but with a comparatively high number of parallel tasks, where the parallelization is of order $O(n^2)$ but the execution will reduces to $O(1)$.

Membrane System for Sub-problem - 3

The third sub-problem is the final step in getting the rank values. This consists of a single membrane. This membrane has n programs and each program consists of operations calculate the weighted sum of the values given by the users for each item. The variables $x_{1,1},\ldots,z_{n,1}$ are assigned the normalized values of the attributes corresponding to each item given by the user. It starts from item 1 (row-wise), for each item till the n^{th} (n^{th} row) item. The variables $w_{1,1},\ldots,w_{n,1}$ represents the set of weights that have been calculated in sub-problem 1. The variables $r_{1,1},\ldots,r_{n,1}$ represents the final values whose descending order determines the rank i.e. higher the value, the higher is the rank of the corresponding item.

Role of Enzyme:

This has only a single enzyme that controls the execution. The value of the enzyme must be sufficiently large so that all the programs execute.

$$
\begin{array}{l}
\underline{Sub-problem3-M_1} \\
\qquad\qquad\qquad\qquad x_{1,1}[value],\cdots,x_{n,1}[value] \\
\qquad\qquad\qquad\qquad \cdots \\
\qquad\qquad\qquad\qquad \cdots \\
\qquad\qquad\qquad\qquad z_{1,1}[value],\cdots,z_{n,1}[value] \\
\qquad\qquad\qquad\qquad w_{1,1}[value],\cdots,w_{n,1}[value] \\
\qquad\qquad\qquad\qquad r_{1,1}[0],\cdots,r_{n,1}[0],e_1[l] \\
Pr_{1,1}:\ x_{1,1}\times w_{1,1}+\cdots+x_{n,1}\times w_{n,1}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid r_{1,1} \\
\qquad\qquad\qquad\qquad \cdots \\
\qquad\qquad\qquad\qquad \cdots \\
Pr_{n,1}:\ z_{1,1}\times w_{1,1}+\cdots+z_{n,1}\times w_{n,1}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid r_{n,1}
\end{array}
$$

Figure 3.4 Membrane System for Sub-problem 3

Further Parallelization of Sub-problem 3

Same as in the case in sub-problem 2, the above mentioned method consists of comparatively less number of programs. This sub-problem can be parallelized further where the program $Pr_{1,1}$ *to* $Pr_{n,1}$ can be divided as in figure 3.5. By doing so, the number rules increase by factor n for each program. Thus, maximal parallelism is achieved, with a com-

paratively high number of parallel tasks, where the parallelization is of order $O(n^2)$ but the execution reduces to order $O(1)$.

$\underline{Sub-problem3-M_1}$

$$x_{1,1}[value], \cdots, x_{n,1}[value]$$
$$\vdots$$
$$z_{1,1}[value], \cdots, z_{n,1}[value]$$
$$w_{1,1}[value], \cdots, w_{n,1}[value], \quad fin_{1,1}[0], \cdots, fin_{m,1}[0]\, e_1[l] \ldots e_{2n}[]$$

$$Pr_{1,1} : x_{1,1} \div max_{1,1} \times w_{1,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid fin_{1,1}$$
$$\vdots$$
$$Pr_{n,1} : x_{n,1} \div max_{n,1} \times w_{n,1} \ \langle e_n \rightarrow \rangle \ 1 \mid fin_{1,1}$$
$$\vdots$$
$$Pr_{(n-1)n+1,1} : z_{1,1} \div max_{1,1} \times w_{1,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid fin_{m,1}$$
$$\vdots$$
$$Pr_{n*m+1,1} : z_{n,1} \div max_{n,1} \times w_{n,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid fin_{m,1}$$
$$Pr_{1,1} : min_{1,1} \div x_{1,1} \times w_{1,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid fin_{1,1}$$
$$\vdots$$
$$Pr_{n,1} : max_{n,1} \div x_{n,1} \times w_{n,1} \ \langle e_n \rightarrow \rangle \ 1 \mid fin_{1,1}$$
$$\vdots$$
$$Pr_{(n-1)n+1,1} : min_{1,1} \div z_{1,1} \times w_{1,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid fin_{m,1}$$
$$\vdots$$
$$Pr_{n*m+1,1} : min_{n,1} \div z_{n,1} \times w_{n,1} \ \langle e_1 \rightarrow \rangle \ 1 \mid fin_{m,1}$$

Figure 3.5 Membrane System for Sub-problem 3 - Parallelized Further

3.2.3 Implementation and Results

The ENPS-IAHP has been implemented and analysed for increasing number of attributes against execution time. Implementation of the method is necessary to establish a working proof for the proposed model based on ENPS. For this purpose, a simulation tool named PeP (Enzymatic) Numerical P System simulator designed by Florea and Buiu (2017) has been used. Another tool also for this is a Java based simulator (based on P-Lingua) developed by Manuel Garcia Quismondo in 2011 (García Quismondo et al., 2012a; García-Quismondo et al., 2009). Nine alternatives have been ranked for 3,5,7,9 attributes for Normal and ENP System, to obtain the corresponding ranks (Figures 3.6 3.7, 3.8, 3.9). This shows that the results are same as the results obtained by Normal Sequential IAHP.

The ENPS-IAHP model is primarily proposed for consistent weight calculation, in case, if weights are not directly provided. Further, the chapter proposes ENPS-based

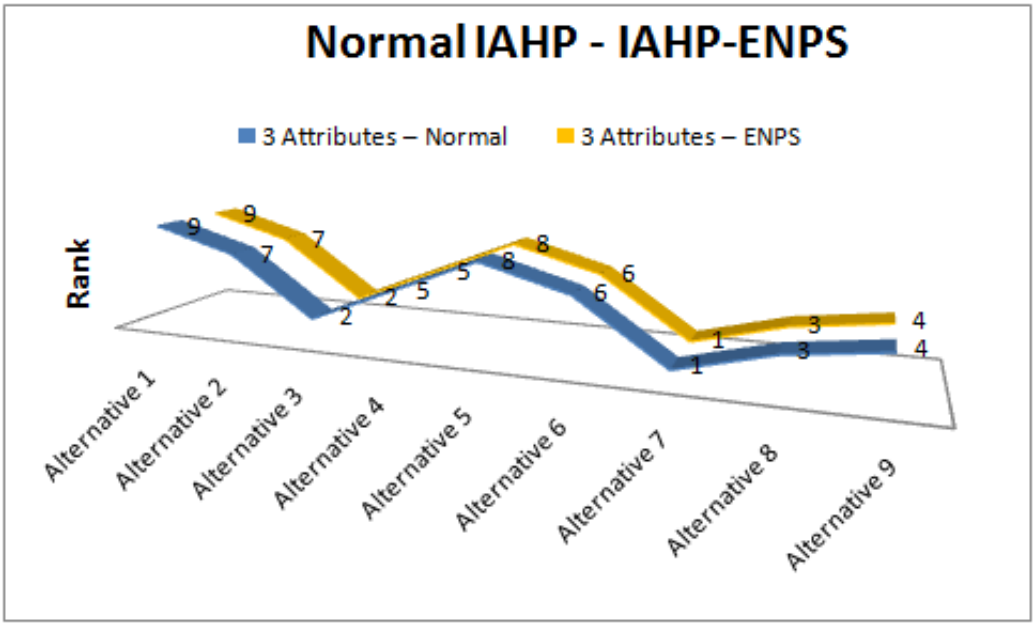

Figure 3.6 IAHP and ENPS-IAHP for 3 attributes

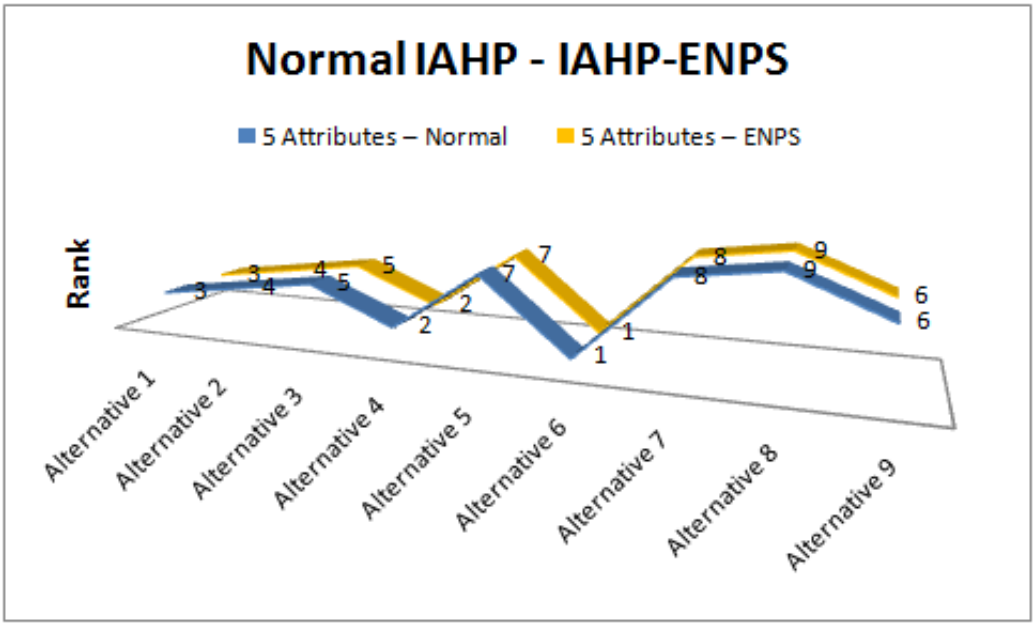

Figure 3.7 IAHP and ENPS-IAHP for 5 attributes

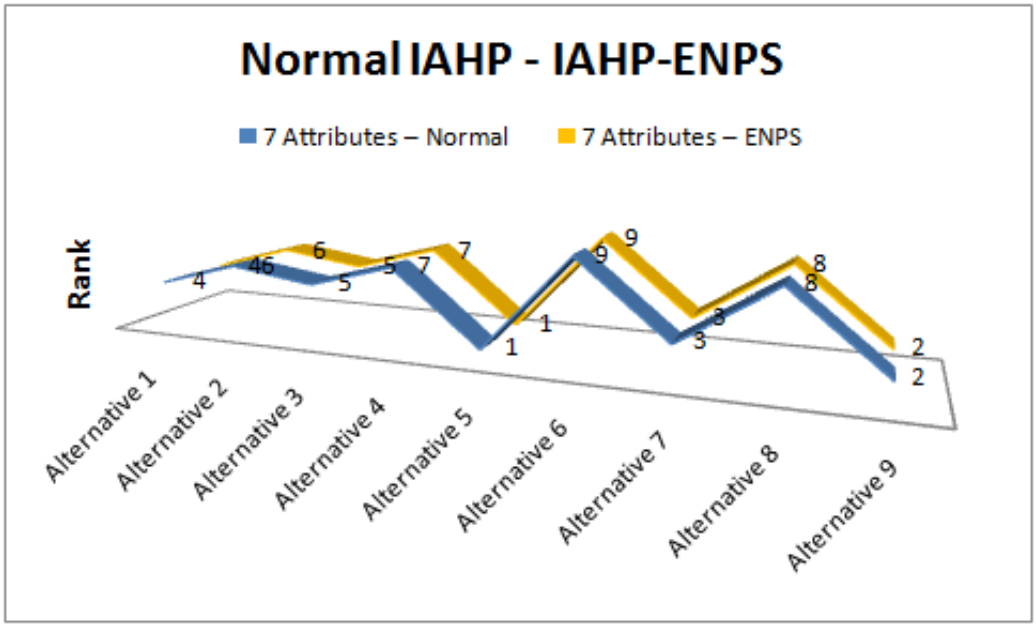

Figure 3.8 IAHP and ENPS-IAHP for 7 attributes

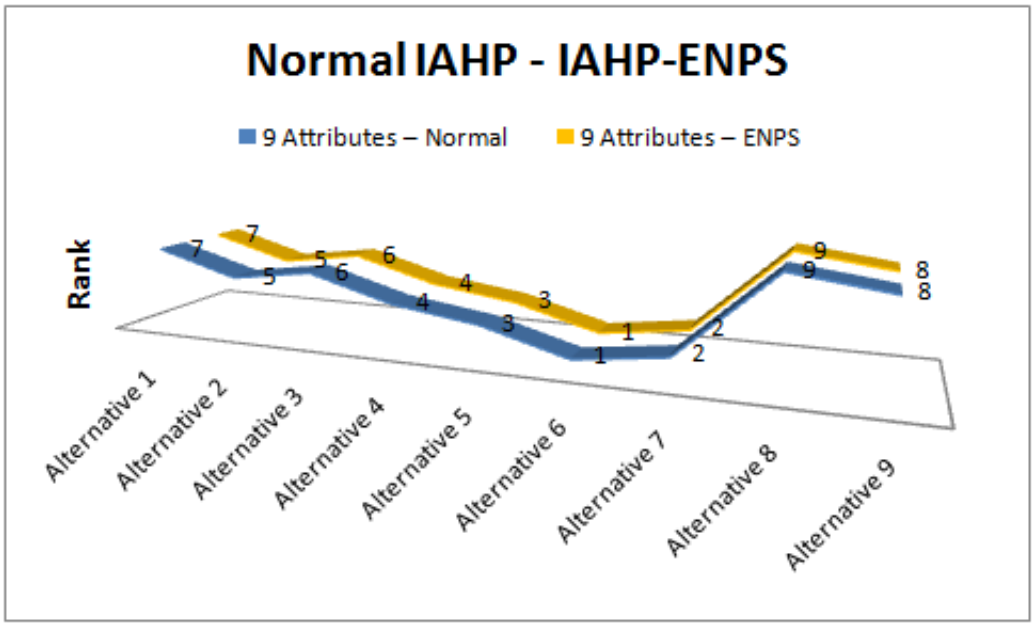

Figure 3.9 IAHP and ENPS-IAHP for 9 attributes

MCDM method called as ENPS-IPROMETHEE.

3.3 Enzymatic Numerical P System - Improved Preference Ranking Organization Method for Enrichment Evaluation (ENPS-IPROMETHEE)

ENPS-IPROMETHEE, the proposed membrane based model has a sequential equivalent, elaborated in this study for a better understanding of the process. Membrane computing can be used to solve several problems, whose solutions are often realized using membrane algorithms, which are strictly or loosely based on membrane computing paradigm. A membrane based algorithm designed, is strictly based on ENPS, a variant of Numerical P System. The aim is to design a parallel MCDM approach. Decision making is a ubiquitous process that prevails in many important domains such as Engineering, Economics and Management. Decision making itself is a vast domain and has several important sets of approaches. MCDM methods is one such set of methods which is used when deciding the best outcome, given several choices with multiple attribute for each choice. These MCDM methods can be classified into several categories based on the procedure followed; Outranking based methods, AHP/ANP based approaches, MAUT based approaches, Simple additive weighing based approaches etc. (Sun et al., 2014).

There are several outranking based methods available and Preference Ranking Organization Method for Enrichment Evaluation (PROMETHEE) is one of the important approaches. It has been developed by Brans (1982) and later an improved version called as Improved PROMETHEE has been developed by Rao (2007). IPROMETHEE is used for ranking a set of alternatives, given the values corresponding to each attribute (property), with the weight of the attribute. Ranking involves listing best solution in order. IPROMETHEE involves pair wise comparison between all the values in a certain order, with the help of preference function. This results in some intermediate values and are used for net flow calculations based on which final ranks are obtained. For Membrane-based design the ENPS model is used as the base for the approach. The ENPS model primarily uses numerical value based variables and this allows the performance of numerical calcu-

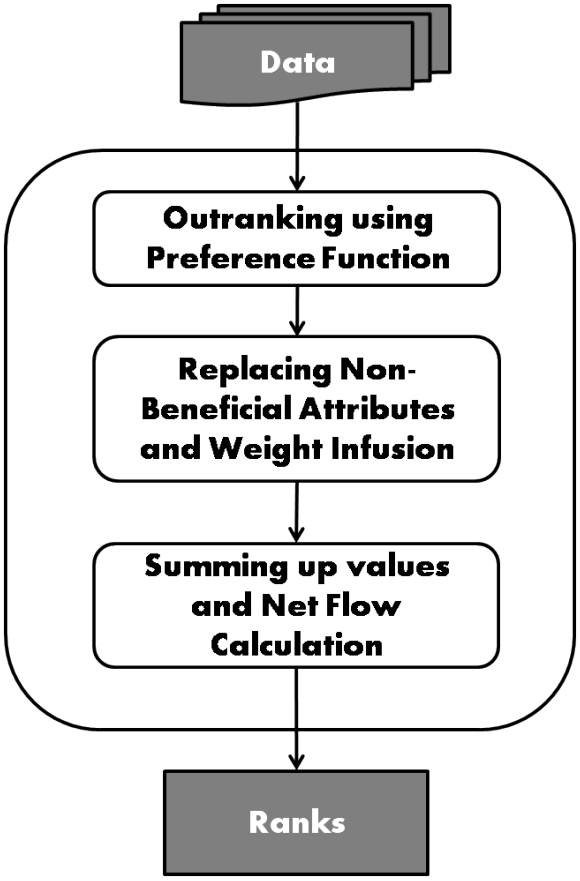

Figure 3.10 Sequential Equivalent

lations pertaining to this method. The details of the Membrane System for the proposed work have been elaborated in the following sections.

3.3.1 Sequential Equivalent

The sequential structure is followed as in figure 3.10. Structurally and in several computational aspects this method is different from IPROMETHEE and it is completely parallel method unlike IPROMETHEE, which is sequential (Rao, 2007). There are three primary steps involved for ranking through the proposed (ENPS-IPROMETHEE) method. Given all the numerical attributes, i.e. the alternatives and attributes with their corresponding weight values, here the following steps are followed. The individual formula for each step are as used in standard IPROMETHEE (Rao, 2007). The first step involves considering each attribute separately and performing pair wise comparison as in the equation 3.3.

Pairwise comparison involves comparing of every single value (alternative) of a par-

ticular attribute with the other alternatives attribute. In this study, it particularly involves working according to the preference function as defined in equation 3.3. The results of comparison of two pairwise value completely depends on preference function. K_j denotes a non-decreasing function, which is over difference d calculated between values 'a1' and 'a2', with 'a2' as base value. There are several different functions for calculating the values of K_j Rao (2007). It is at the discretion of the decision maker to use a suitable preference function according to the problem. The 'usual' function is considered, whose definition is as given in equation 3.4. The 'usual' preference function gives $n*n$ values, each of which is binary in nature i.e. each value is either 0 or 1. The comparison results in a set of $n*n$ values.

The next step considers differentiating between beneficial attributes and non-beneficial attribute and also infusing weights into each of the resultant value, accordingly. For the beneficial attributes the higher the attribute value the better is the impact, while it is reverse for the non-beneficial attribute i.e. the lower the value the better is the impact. The step of differentiating beneficial attribute from non-beneficial attributes is done in this step unlike IPROMETHEE (Rao, 2007) where, preference function itself take cares of it with few additional conditions. The values of the attributes obtained after the first step can be considering as beneficial attribute. To convert the result according to non-beneficial attribute, theoretically a transpose of the resultant $n*n$ matrix is taken. This results in the same value as in for a non-beneficial attribute.

$$P_{j,a1,a2} = K_j[b_j(a1) - b_j(a2)] \tag{3.3}$$

$$where \;\; 0 \leq P_{j,a1,a2} \leq 1$$

$$K_j(d_j) = \begin{cases} 1 & \text{if } d_j > 0 \\ 0 & \text{else } d_j \leq 0 \end{cases} \tag{3.4}$$

$$I_{a1,a2} = \sum_{j=1}^{M} w_j P_{j,a1,a2} \tag{3.5}$$

$$\theta^{+}(a) = \sum_{x \in A} I_{Xa} \tag{3.6}$$

$$\theta^{-}(a) = \sum_{x \in A} I_{aX} \tag{3.7}$$

$$\theta(a) = \theta^{+}(a) - \theta^{-}(a) \tag{3.8}$$

The reason behind changing the sequence and method is purely based on structural properties of ENPS system whose benefit can be understood in the next section, which elaborates on the exact algorithm of ENPS-IPROMETHEE. Based on beneficial and non-beneficial values the resultant value obtained are multiplied with the corresponding weights of the attributes and the final set of values for each attribute with considered weight, is obtained. The resultant value I (as shown in equation 3.5) is the intensity which later is used for determining the domination of function.

The next step consists of combining all the values $(n * n)$ into single $n * n$ matrix. There is no separate step needed for doing this, as the ENPS structure allows passing of the values directly to the concerned variable and it automatically adds all the passed values, without explicitly having production function doing it; the next section elaborates on this. Once this is done the final step is taken up where net flow is calculated. The outflow (leaving flow) $(\theta^{+}(a))$ measures the dominance of value over the other values (as in equation 3.6). The inflow $(\theta^{-}(a))$ measures the collective dominance of other values over the considered value (Rao, 2007). The difference between this outflow and the inflow gives the net flow $(\theta(a))$. This net flow is obtained for each alternative as in equation 3.8. This when sorted in descending order gives the ranks of the alternatives. The next section gives details of ENPS-IPROMETHEE with the actual strict ENPS structure.

3.3.2 ENPS-IPROMETHEE Membrane Structure

ENPS-IPROMETHEE is a single membrane system solution i.e. the problem involves only one membrane system where there is a single skin membrane with several membranes contained in it (in figure 3.12). The problem is divided into three primary components; calculation of outranking values, differentiating beneficial and non-beneficial attributes

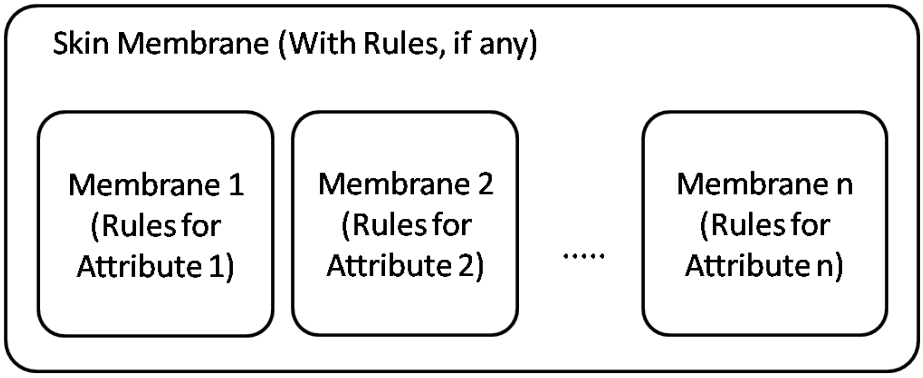

Figure 3.12 ENPS-IPROMETHEE Structure

and infusing weights for each attribute with combining the calculated values, and finally calculating the net flow to get the ranks.

$$\underline{Skin - Membrane}$$

$$x1_{1,1}[0], \ldots, xn_{1,1}[0]$$
$$\ldots$$
$$x1_{2,1}[0], \ldots, xn_{n,1}[0]$$
$$i_{1,1}[0], \ldots, i_{n,1}[0], e_{1,1}[0], e_{2,1}[0]$$

$$Pr_{1,1} : (x1_{1,1} + \ldots + xn_{1,1}) - (x1_{1,1} + \ldots + x1_{n,1}) \ \langle e_{2,1} \rightarrow \rangle \ 1 \mid i_{1,1}$$
$$\ldots$$
$$Pr_{n,1} : (x1_{2,1} + \ldots + xn_{1,1}) - (x2_{1,1} + \ldots + x2_{n,1}) \ \langle e_{2,1} \rightarrow \rangle \ 1 \mid i_{n,1}$$
$$Pr_{n+1,1} : 1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid e_{2,1}$$

Figure 3.11 Skin Membrane

Cycles:

Each cycle is a step where all the programs are executed together, once, parallely. In the first cycle the outranking values are calculated according to the 'usual' function and the corresponding resultant values are passed on to be used in the next cycle. The next cycle involves infusion of weights to the attributes and further differentiation between beneficial and non-beneficial attributes, thereby accordingly changing the values. The resultant values are finally passed, and are to be used in the next cycle where the inflow and outflow is calculated and their difference is obtained. Thus, on the whole, only three cycles are followed for any number of attributes and any number of alternatives. Theoretically,

considering parallelism, the time complexity can be $O(5)$ i.e. $cO(1)$, a constant.

$\underline{Child - Membrane - k}$

$$
\begin{gathered}
x_{1,k}[0], \ldots, x_{n,k}[0], \\
x1_{1,k}[0], \ldots, xn_{1,k}[0], \\
\cdots \\
x1_{n,k}[0], \ldots, xn_{n,k}[0], \\
i_{1,k}[0], \ldots, i_{n,k}[0], \\
x_{1,k}[0], \ldots, x_{n,k}[0], e_{1,k}[0], \ldots, e_{n,k}[0], e_{11,k}, e_{12,k}
\end{gathered}
$$

$$
\begin{gathered}
Pr_{1,k} : (x_{1,k}/x_{1,k}) \ \langle e_{1,k} \rightarrow \rangle \ 1 \mid x1_{1,1} \\
\cdots \\
Pr_{n,k} : (x_{1,k}/x_{1,k}) \ \langle e_{n,k} \rightarrow \rangle \ 1 \mid xn_{1,k} \\
\cdots \\
\cdots \\
Pr_{n^2-n+1,k} : (x_{n,k}/x_{n,k}) \ \langle e_{1,k} \rightarrow \rangle \ 1 \mid x1_{n,k} \\
\cdots \\
Pr_{n^2,k} : (x_{n,k}/x_{n,k}) \ \langle e_{n,k} \rightarrow \rangle \ 1 \mid xn_{n,k} \\
Pr_{n^2+1,k} : (x1_{1,k} * w_{1,k}) \ \langle e_{11,k} \rightarrow \rangle \ 1 \mid x1_{1,1} \\
\cdots \\
Pr_{n^+n,k} : (xn_{1,k} * w_{1,k}) \ \langle e_{11,k} \rightarrow \rangle \ 1 \mid x1_{n,1} \\
\cdots \\
\cdots \\
Pr_{2n^2-n+1,k} : (x1_{n,1} * w_{1,k}) \ \langle e_{11,k} \rightarrow \rangle \ 1 \mid xn_{1,k} \\
\cdots \\
Pr_{2n^2,k} : (xn_{n,k} * w_{1,k}) \ \langle e_{11,k} \rightarrow \rangle \ 1 \mid xn_{n,1} \\
Pr_{2n^2+1,k} : (x1_{1,k} * w_{1,k}) \ \langle e_{12,k} \rightarrow \rangle \ 1 \mid x1_{1,1} \\
\cdots \\
Pr_{2n^2+n,k} : (xn_{1,k} * w_{1,k}) \ \langle e_{12,k} \rightarrow \rangle \ 1 \mid x1_{n,1} \\
\cdots \\
\cdots \\
Pr_{3n^2-n+1,k} : (x1_{n,k} * w_{1,k}) \ \langle e_{12,k} \rightarrow \rangle \ 1 \mid xn_{1,1} \\
\cdots \\
Pr_{3n^2,k} : (xn_{n,k} * w_{1,k}) \ \langle e_{12,k} \rightarrow \rangle \ 1 \mid xn_{n,1} \\
Pr_{3n^2+1,k} : (x1_{1,k} * e_{1,k} * \ldots * e_{n,k}) \ \langle e_{1,k} \rightarrow \rangle \ 1 \mid e_{1,k} + \ldots + 1 \mid e_{n,k} \\
Pr_{3n^2+2,k} : (e_{11,k} - 2 * e_{11,k}) \ \langle e_{1,k} \rightarrow \rangle \ 1 \mid e_{11,k} \\
Pr_{3n^2+3,k} : (e_{12,k} - 2 * e_{12,k}) \ \langle e_{1,k} \rightarrow \rangle \ 1 \mid e_{12,k}
\end{gathered}
$$

Figure 3.13 Core Membrane (For a single attribute)

Membranes

As in figure 3.12 there are two important partitions, according to rules; skin membrane and child membranes, present inside the skin membrane. The child membrane can be of any number, depending on the number of attributes, in this case. Each created membrane

caters for one attribute and the calculations pertaining to each attribute are carried out in each of them.

Child membrane:

The child membrane corresponding to k^{th} attribute is shown in figure 3.13. In total there are $3n^2+3$ programs for each membrane, where n is the number of alternatives. The first n^2 programs from 1 to n^2 are used to assign the comparison values obtained according to the preference function given. 'Usual' preference function is considered here. This first set is controlled by using n enzymes (equal to the number of alternatives). Based on the type of attributes, the execution of programs are controlled.

The next $2*n^2$ programs are considered for weight calculations according to the beneficial or non-beneficial attributes. The first set (n^2) of programs are for beneficial attributes and the next set of programs are for non-beneficial attributes. The programs from n^2+1 to $2n^2$ are executed only if the attributes are non-beneficial and the programs from $2n^2+1$ to $3n^2$ are executed only if the attribute is beneficial. The separate operations for beneficial and non-beneficial attributes is controlled by two enzymes, $e_{11,1}$ and $e_{12,1}$. These two enzymes are separate for each membrane and they are assigned accordingly before the start of execution, so that, while the membranes are executed the calculation is automatically done according to the values assigned. While these calculations are done normally at the side of the production function, the property of membrane computing allows the calculated values to be passed directly to the variables in other membranes in a parallel manner. The values, after they have been processed, are passed on to the variables of skin membranes. The set of operations are replicated for each attributes and all of them are executed in an all-parallel manner.

Skin membrane:

A set of operations happen at the skin membrane, as shown in figure 3.11. The outranking values with weights passed to the skin membrane by all the other membranes, combine because of the ENPS property of gathering. In the next step, the net flow is calculated. The

programs 1 to n are used for this purpose, as the final attribute value are already present before this step. A single production function adds up the inflow and subtracts total outflow from the added value. The final values based on which the ranks are obtained, are stored from $i_{1,1}$ to $i_{n,1}$. The execution of these programs is controlled by two enzymes $e_{1,1}$ and $e_{2,1}$ in the third cycle. Finally, the values available in variables from i_1 to i_n give ranks when sorted in descending order.

3.3.3 Case Studies for ENPS-IPROMETHEE with Implementation

There are two required values for calculating the ranks of a given data. The weights of the values, and the actual values of the attributes corresponding to each alternative. In order to ascertain the working of the purposed model, two case studies are done. These case studies use existing standard datasets for evaluating the proposed method. Two datasets are considered (for each case study). The first being a dataset of materials with seven attributes that are to be selected for a cryogenic tank, originally given by Farag (2007). The properties of the material has been given i.e. their category of whether they are beneficial or non-beneficial has been mentioned. This is one of the most commonly used dataset for showing effectiveness of a MCDM algorithm. Further in the second case study, a dataset for selection of green material for sustainability is used. It has been initially used by Zhang et al. (2017b). The number of attributes are more than the previous dataset (case study 1) that come up to 14 and it has five materials.

For implementing the designed model a Python based ENPS simulator, PeP 3.0, completely based on Python developed by Florea and Buiu (2018) is used. Further, as a GPU alternative for PeP, GPUPeP has been developed and used. The final model designed, requires an automatic file generator working over PeP and GPUPeP simulator which can run the ENPS-IPROMETHEE model (directly over these tools) to obtain the results. The generator TP-Generator is developed, based on Python 3.0 and is tailored with the simulators to give the final ranks of the items, given the weights and values of the attributes. Further, AHP has also been used for finalizing the weights, in case, the user has only relative importance of the attributes, other than the actual weight.

Material Number	Material
Material 1	Al 2024-T6
Material 2	Al 5052-O
Material 3	SS 301-FH
Material 4	SS310-3AH
Material 5	Ti-6Al-4 V
Material 6	Inconel 718
Material 7	70Cu-30Zn

Table 3.2 Details of the materials used (Farag, 2007)

Case Study I: Material Selection for Cryogenic Tank

The demonstration of the accuracy of the approach, ENPS-IPROMETHEE is done through a standard material selection problem. The material selection problem is a process in which a suitable material is selected, based on several properties (attributes) of the material. Each and every attribute has its own importance which can be shown by having assigned weights. The data used is given in Farag (2007), which is for material selection for cryogenic tank.

This data is considered because of its variety (range) which is apt for testing a general MCDM problem. It has been used by several researchers, like Dehghan-Manshadi et al. (2007) and Rao (2007), for testing several decision making approaches like Improved Analytic Hierarchy Process (IAHP), Improved Technique of Order Preference Similarity to the Ideal Solution (ITOPSIS), IPROMETHEE etc. This table is used in order to demonstrate the approach of this research. The data being used, consists of about seven materials and seven attributes. All the seven attributes have weight that has been calculated using part of IAHP. Preference is assigned for each attribute, in comparison to other attribute, based on the general perception. The proposed P System based ENPS-IPROMETHEE method is used for ranking the alternatives.

The details of the attributes are given in table 3.4 with the weights. Weights are calcu-

	T	YS	YM	D	TE	TC	SH
T	1	2	6	1.2	1.5	6	6
YS	0.5	1	3	0.5	0.75	3	3
YM	0.166	0.333	1	0.2	0.25	1	1
D	0.833	2	5	1	1.33	5	5
TE	0.667	1.33	4	0.75	1	4	4
TC	0.166	0.33	1	0.2	0.25	1	1
SH	0.166	0.33	1	0.2	0.25	1	1

Table 3.3 Preference matrix for the attributes

lated using the AHP which inturn uses a preference matrix for doing so. The preference matrix is usually filled by the users who have a general perception about the attributes. Though according to standard IAHP Rao (2007), whole numbers ranging from 1-9 are considered, for preference matrix. In this work, real numbers are considered to denote the relation among the attribute to avoid greater discordance between the weights. In addition to that, this consideration increases the precision if the user is very clear with his/her perception.

This is two dimensional matrix, where the preference of every attribute is plotted against each other as shown in table 3.3 and then IAHP (Rao, 2007) method is used for calculating weights through the given preference matrix. The weights obtained have been considered as presented in table 3.5. After the weights are obtained a consistency check is supposed to be made (which is a part of IAHP itself) (Rao, 2007). If the user data is found to be consistent then the weights are properly assigned, otherwise the IAHP process is repeated all over again till consistent weights are obtained.

After the weights are obtained, it is passed on with the other values of alternatives and attributes, through the membrane generator (P System Generator) which generates the required membrane according to the format. The resultant membranes are executed using PeP and GPUPeP (GPU based membrane execution simulator).
The rank of the materials for ENPS-IPROMETHEE is as follows:

Material Selection Attributes	Unit/ Remark	Beneficial/ Non-Beneficial	Weight
Toughness index (TI)	based on UTS, Yield Strength (YS) and Ductility (e) at -196 °C = (UTS + YS)e/2	Beneficial	0.285
Yield Strength (YS)	MPa	Beneficial	0.138
Young's Modulus (YM)	GPa	Beneficial	0.047
Density (D)	g/cm^3	Non-Beneficial	0.246
Thermal Expansion (TE)	10^{-6}/°C	Non-Beneficial	0.188
Thermal Conductivity (TC)	cal/cm^2 /cm/°C/s	Non-Beneficial	0.047
Specific Heat (SH)	cal/g/°C	Non-Beneficial	0.047

Table 3.4 Attributes and their details (Farag, 2007)

Materials	TI	YS	YM	D	TE	TC	SH
Material 1	75.5	420	74.2	2.8	21.4	0.37	0.16
Material 2	95	91	70	2.68	22.1	0.33	0.16
Material 3	770	1365	189	7.9	16.9	0.04	0.08
Material 4	187	1120	210	7.9	14.4	0.03	0.08
Material 5	179	875	112	4.43	9.4	0.016	0.09
Material 6	239	1190	217	8.51	11.5	0.31	0.07
Material 7	273	200	112	8.53	19.9	0.29	0.06

Table 3.5 Materials and their corresponding attribute values (Reproduced with permission) (Farag, 2007)

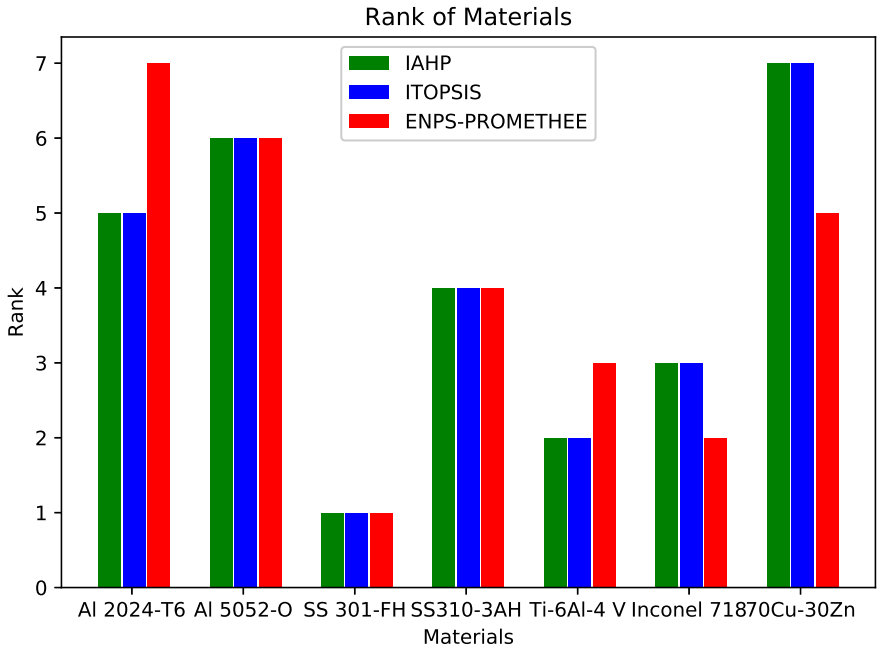

Figure 3.14 Final ranking

SS 301-FH → Inconel 718 → Ti-6Al-4V → SS310-3AH → 70Cu-30Zn → Al 5052-O → Al 2024-T6.

Results and Analysis

Figure 3.14 shows the comparison of ENPS-IPROMETHEE method with IAHP and ITOPSIS method. After implementation the results obtained are analysed for sensitivity.

Sensitivity Analysis:

Sensitivity analysis is a process of studying the behaviour (results) of the algorithm by interchanging the existing weights for the attributes. When weights are interchanged the algorithm may give different results for every change in weight. This change in ranks is plotted and analysed. Thus sensitivity of the algorithm to change is analysed. In this work, 20 cases of interchanged weights are considered and corresponding ranks are obtained. These ranks are plotted against the corresponding weight interchange values (Figure 3.15). The results show that there are changes is the ranks obtained as the weights change. But the best rank obtained is for material three in all the cases except three cases. The rank changes are plotted in the graph. Thus, as expected, in this case, ENPS-IPROMETHEE is

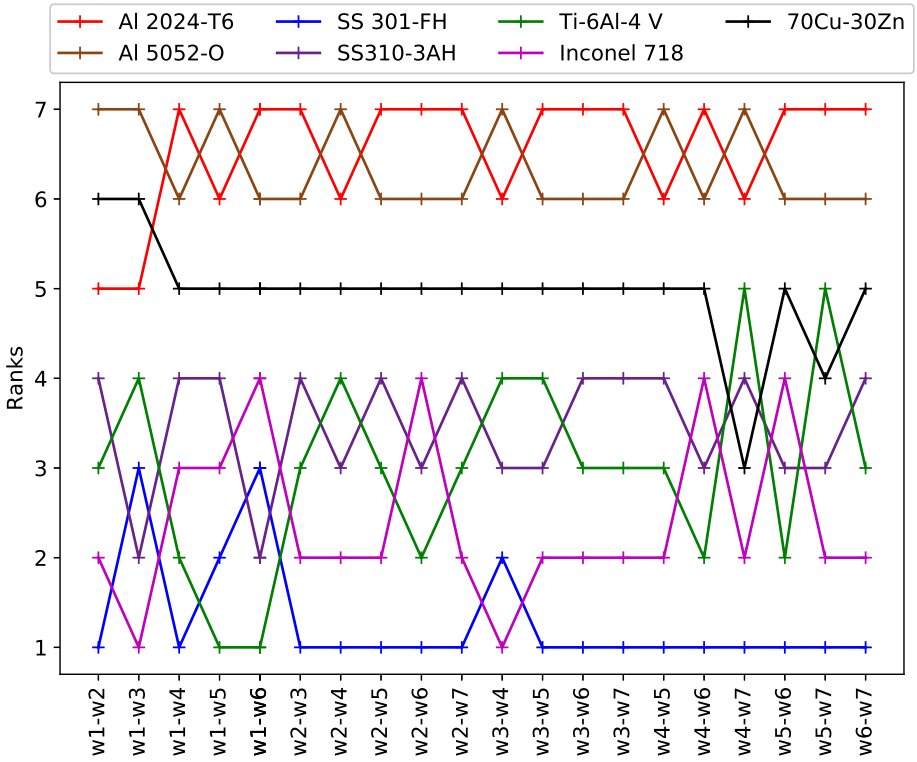

Figure 3.15 Sensitivity analysis for material selection

moderately sensitive to the change in weights.

Case Study II - Green Material Selection for Sustainability

To demonstrate the sensitivity of ENPS-IPROMETHEE, with more attributes, another dataset of a material selection is considered, specifically to select a green material for sustainable development (Zhang et al., 2017b). This dataset has originally been used by Zhang et al. (2017b) for getting the best green material for sustainability. There are, in total, 14 attributes based on which five materials are ranked the details are shown in table 3.7. As per the procedure, the proposed ENPS-IPROMETHEE model is applied, followed by a comparison of the results with IAHP and ITOPSIS method. Further Sensitivity analysis is done over the results.

The weights given by Zhang et al. (2017b) have directly been used for analysing the present approach, depicted in table 3.6. The table 3.7 represents the actual data considered. The proposed method is applied and implemented and the rank is obtained over the given

Attribute Number	**Attribute Name**	**Beneficial / Non-Beneficial**	**Weight**
C1	Initial cost	Non-Beneficial	0.052
C2	Maintenance cost	**N**on-Beneficial	0.061
C3	Disposal cost	Non-Beneficial	0.067
C4	Tax contribution	Beneficial	0.052
C5	Energy saving	Beneficial	0.086
C6	Potential for recycling and reuse	Beneficial	0.112
C7	Raw material extraction	Beneficial	0.054
C8	Usage of water	Non-Beneficial	0.092
C9	Density	Non-Beneficial	0.083
C10	CO2 Emission	Beneficial	0.053
C11	Rigidity	Non-Beneficial	0.076
C12	Tensile strength	Beneficial	0.069
C13	Elongation at break	Beneficial	0.074
C14	Tensile modulus	Non-Beneficial	0.069

Table 3.6 Materials and their corresponding attribute values (Zhang et al., 2017b)

Material	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
AL-REM	2	2	3	3	3	3	4	3	3	2.72	50	169	8	25
ABS-INC	2	3	2	3	2	3	3	2	2	1.34	100	90	2	7.9
ABS-LND	3	2	3	2	2	4	2	4	3	1.34	100	90	2	7.9
PU-INC	3	3	2	3	4	2	3	4	3	1.15	60	27	10	4.5
PU-LND	4	4	2	4	4	3	3	3	4	1.15	60	27	10	4.5

Table 3.7 Materials and their corresponding attribute values (Zhang et al., 2017b)

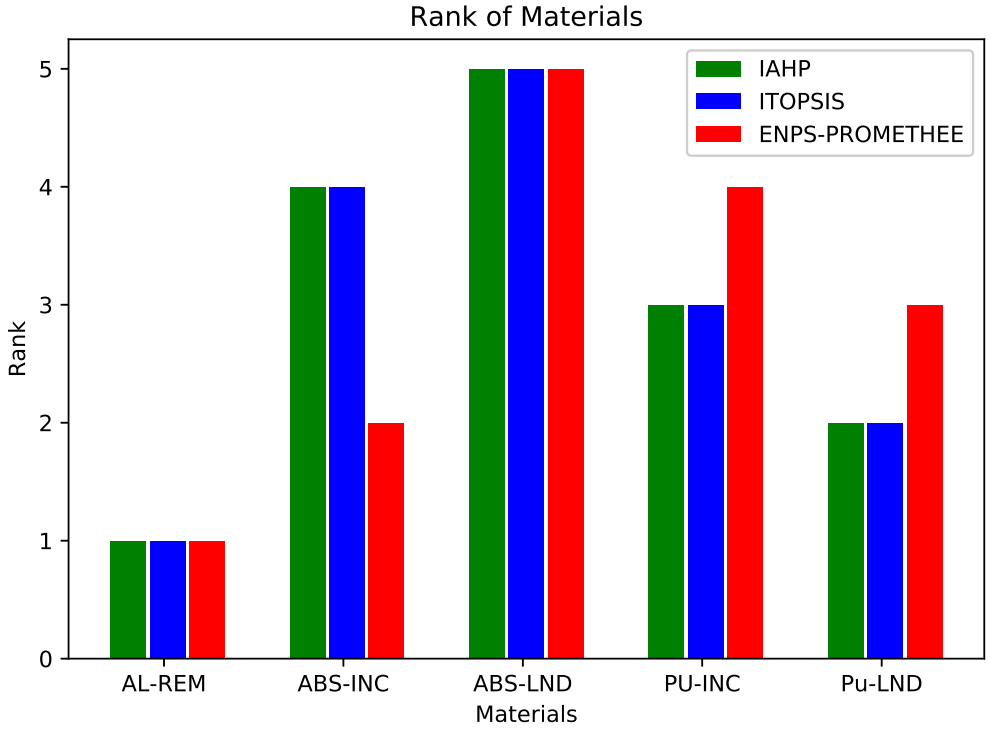

Figure 3.16 Final ranking for green material selection

values as shown below.

AL-REM → ABS-INC → PU-LND → PU-INC → ABS-LND

Results and Analysis

The results are compared with IAHP and ITOPSIS as shown in figure 3.16. Sensitivity analysis is done to ascertain the sensitivity of the proposed method.

Sensitivity Analysis:

The sensitivity analysis gives results as shown in figure 3.17. There are a total of 14

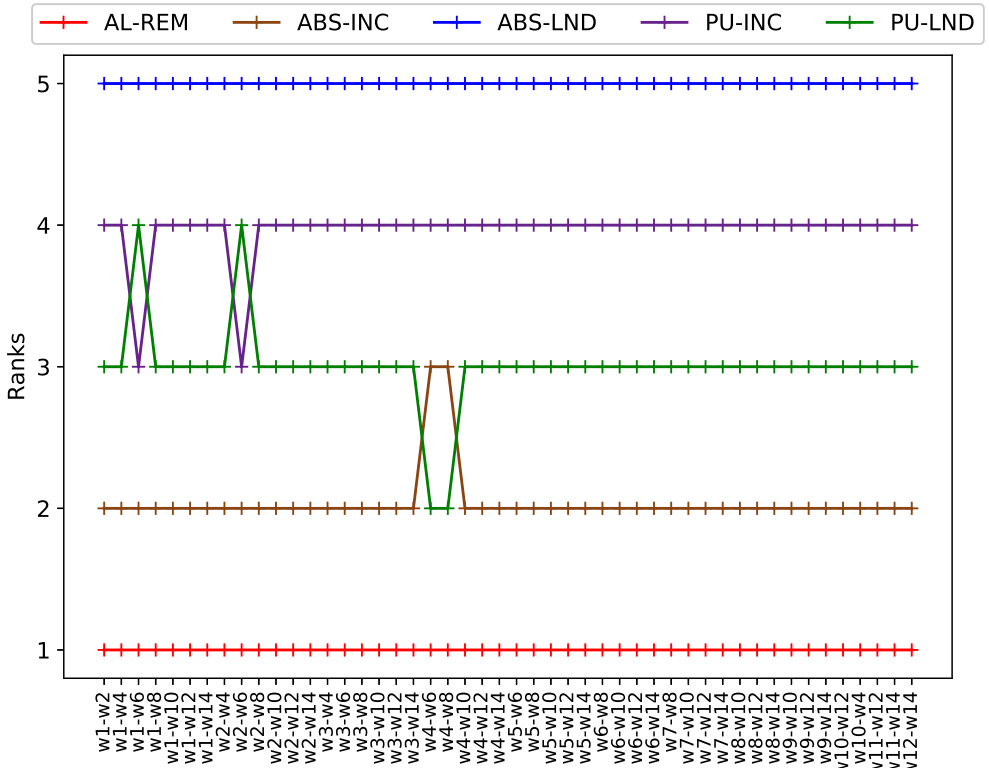

Figure 3.17 Sensitivity analysis for green building selection

attributes, considering all the possible combinations of weight changes can lead to too many inter-changes (91 times). Instead of considering all the changes, total of 48 swaps are considered. The results are indicated in figure 3.17. Out of 48 cases there are only four cases where there has been a change, which can be attributed to several reasons including increase in attributes or reduction in difference of weights. Thus the overall method is moderately sensitive for this particular dataset.

3.4 Summary

There are two primary contributions of this chapter. One is a weight calculation method based on Enzymatic Numerical P System (ENPS) and Improved Analytic Hierarchy Process (IAHP). This method named ENPS-IAHP is used to calculate the weight given the user preference matrix consisting of preference of one attribute over all other. This can also be used for ranking a set of items but it is primarily proposed for weight calculation. The second contribution is a ranking method for service (item) selection . This method

again is structurally based on ENPS and is inspired by IPROMETHEE (Rao, 2007). This method is implemented and results are analysed which show it to be moderately sensitive to weight changes.

Chapter 4

TOOLS FOR ENZYMATIC NUMERICAL P SYSTEM

There are two important tools developed as part of the project and this chapter elaborates on them. Both the developed tools are for Enzymatic Numerical P System (ENPS). The first tool is Multi-ENPS tool that allows execution of multiple membrane systems and additionally supports two simulators (Python-based and Java-based). The second tool is a GPU-based simulator for ENPS model named GPUPeP. Finally, both the tools are tested and results show the tools to be working as intended.

4.1 Multi-ENPS Simulator Support Tool with Automatic File Interconversion and Multi-membrane Execution

Though there are several tools available (as discussed in section 2.1), there is no tool which supports automatic multiple membrane execution and a tool which allows interchangeable execution between the ENPS simulators. This tool aims to solve both these problems of multiple membrane execution with value passing and interchangeable execution of tools thus enabling interoperability between them.

4.1.1 Introduction

There are several simulators developed to simulate ENPS. The input format for the tools are primarily XML or PeP (Python-based customized format). The first tool proposed here helps inter-conversion between PeP and XML file formats and also allows to execute

```
num_ps = {
    # membrane names (labels)
    H = {m1};

    structure = [m1 ]m1;

    # membrane 1
    m1 = {
        var = {x_1_1, x_2_1, x_3_1, x_4_1, g_1_1}; # variables used in the production function
        E = {e_1_1}; # set of enzyme variables

        pr = {(x_1_1 * x_2_1 * x_3_1) [e_1_1 -> ] 1|g_1_1 + 1|x_1_1 + 1|x_2_1};
        pr = {x_3_1 * 0.015 [e_1_1->] 1|x_3_1};

        var0 = (1, 0.11, 5, 0.8, 9); # initial values for variables x_1_1, x_2_1, x_3_1
        E0  = (40000000); # initial values for enzymes e_1_1, e_2_1
    };
}
```

Figure 4.1 Sample PeP File (Florea and Buiu, 2017, 2018)

multiple membranes using either of the tool (Java-based or Python-based) with automated dependent variable transfer. The primary simulator used for PeP file as input, is PeP simulator and for XML it is JavaENPS (Garcıa-Quismondo, 2013).

PeP file format

The basic template of an ENPS as a PeP file is as shown in figure 4.1.

where:

- *num_ps* : the name of the P system
- H: a list of membrane names
- *structure* : describes the structure of the system.
- $m1 = \{ \ldots \}$: the definition of membrane $m1$. Note the name of the membrane is the same as the one defined in H
- $var = \{\ldots\}$: a comma separated list of P objects that are part of this membrane
- $pr = \{\ldots\}$: the definition of a program.
  - The *pr* keyword is the same for all programs.
  - The right arrow ->is used to separate the production function (left-side) from the distribution function (repartition protocol) (right-side)

- $var0 = (\ldots)$: a comma separated list of initial P object values, specified in the same order as that used for var
- comments start with #
- code blocks are delimited using { } and are used for, $num\_ps, H, m1, var$ and Pr and lists of numeric constants are delimited using ( ) and are used mainly for $var0$
- $E = \{\ldots\}$: a comma separated list of P objects that are the enzymes of this membrane
- $E0 = (\ldots)$: a comma separated list of initial enzyme P object values, specified in the same order as that used for E

XML File format

The basic template of an ENPS, represented as an XML file is as shown in figure 4.2:

where *membraneSystem*: this tag contains the entire structure of ENPS *membrane*: Each membrane tag contains all the relevant attributes and programs of that membrane in the form of various sub-tags, namely, region, and children.

- The region tag contains the characteristics pertaining to that particular membrane.
- The children tag contains the characteristics of the membranes nested inside that membrane, that is, the children of that membrane.

region : This tag gives definition to a membrane. It contains the programs and variables of a membrane. The constituent members of the region tag are:

- *memory*: It contains multiple variable tags under it. The variable tags are of two types:
  - Tags with attributes input and output set to true : These tags collectively represent the list of non-enzyme variables of the membrane. The initial value of the individual variables are stored in the initialValue attribute.

```xml
<membraneSystem type="ENPS" xmlns="http://www.example.org">
<membrane name="m1">
<region>
<memory>
<variable initialValue="1" input="true" output="true">x_1_1</variable>
<variable initialValue="0.11" input="true" output="true">x_2_1</variable>
<variable initialValue="5" input="true" output="true">x_3_1</variable>
<variable initialValue="0.8" input="true" output="true">x_4_1</variable>
<variable initialValue="9" input="true" output="true">g_1_1</variable>
<variable initialValue="40000000" stop="true">e_1_1</variable>
</memory>
<rulesList>
<rule>
<repartitionProtocol>
<repartitionVariable contribution="1">g_1_1</repartitionVariable>
<repartitionVariable contribution="1">x_1_1</repartitionVariable>
<repartitionVariable contribution="1">x_2_1</repartitionVariable>
</repartitionProtocol>
<productionFunction>
<math xmlns="http://www.w3.org/1998/Math/MathML">
<apply>
<times />
<ci>x_1_1</ci>
<ci>x_2_1</ci>
<ci>x_3_1</ci>
</apply>
</math>
</productionFunction>
<enzyme>e_1_1</enzyme>
</rule>
<rule>
<repartitionProtocol>
<repartitionVariable contribution="1">x_3_1</repartitionVariable>
</repartitionProtocol>
<productionFunction>
<math xmlns="http://www.w3.org/1998/Math/MathML">
<apply>
<times />
<cn>0.0150000000000000</cn>
<ci>x_3_1</ci>
</apply>
</math>
</productionFunction>
<enzyme>e_1_1</enzyme>
</rule>
</rulesList>
</region>
<children />
</membrane>
</membraneSystem>
```

Figure 4.2 XML File

- Tags with attribute stop set to true: These tags represent the enzyme variables of the membrane. The value ascertaining the execution of program is stored under the tag initialValue

- *ruleList*: It contains all the programs of a given membrane. Each program is present under a rule tag.

- *children* : This tag contains all the children membranes of the membrane considered as given by the ENPS structure. Each child is stored under a membrane tag.

rule : Each program in an ENPS model is present under and represented by a rule tag. Each rule tag has two (or three) constituents:

- *repartitionProtocol* : This tag encloses the repartition protocol and the proportion of distribution of the production to variables. It has children repartitionVariable tags. Each repartitionVariable tag encloses the variable name and has the attribute contribution which defines its proportion in the production.

- *productionFunction* : This tag encloses the production function part of the program and uses Mathematical Markup Language (MathML) ,to represent it.

- *enzyme* : This tag encloses the enzyme which determines the execution of the program.

4.1.2 Design and Implementation the Tool

The tool developed contains four functional parts as shown in figure 4.3 with all four modules of the tool, their inputs and outputs. The following section contains the explanation of each part along with details of implementation, and diagrams to help understand the components.

Table 4.1 shows mapping between the two formats based on the given examples (figure 4.1 and figure 4.2). Both the examples show the same membrane structure in different formats. The steps are roughly mapped according to converters' execution steps. The difference in process of conversion is because of having two different file structures in

different file formats. Hence, though the sequence of steps is similar, different technologies and approaches are used to handle them. The details of both the converters are given in the following sub-sections.

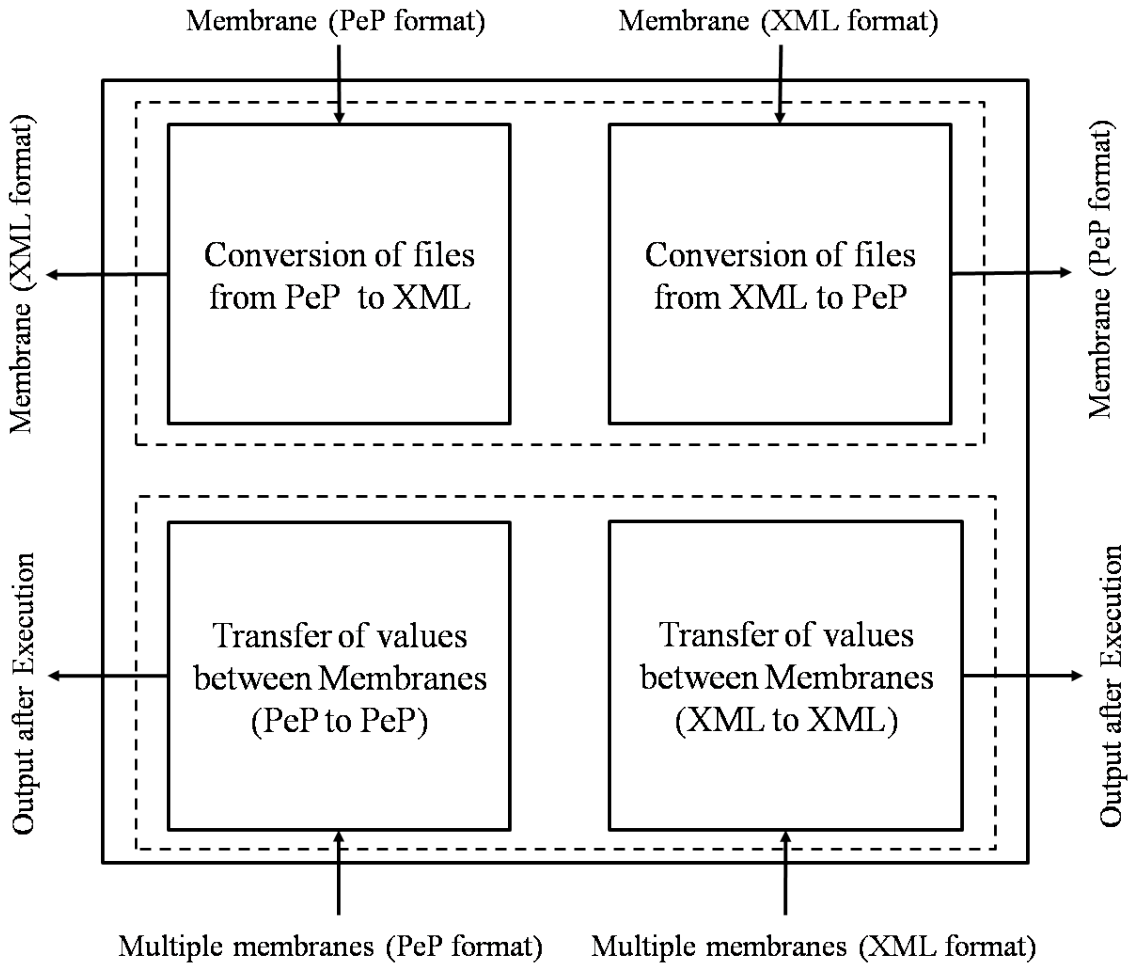

Figure 4.3 Tool Structure

4.1.3 Conversion of files from PeP to XML format

The components of this module are explained as follows:

Analyze input PeP file: The input file is obtained as a command line input and it can be run using the command

Python3 peptoxml.py filename.pep

The developed module analyzes it by processing it line by line. A file named 'out-

Step No.	Component	PeP File (Line No.)	XML File (Line No.)
1	Membrane Structure	3, 5, 8 and 17	1-3 and 47-50
2	Variable and Enzyme	9, 10. 15, 16	4-11
3	Production Function 1	12	12, 13, 19-28, 30
4	Re-partition Protocol 1	12	14-18
5	Enzyme (PF 1)	12	29
6	Production Function 2	13	31, 35-43 and 45
7	Re-partition Protocol 2	13	32-34
8	Enzyme (PF 2)	13	44

Table 4.1 File Structure Mapping and Translation for given Example

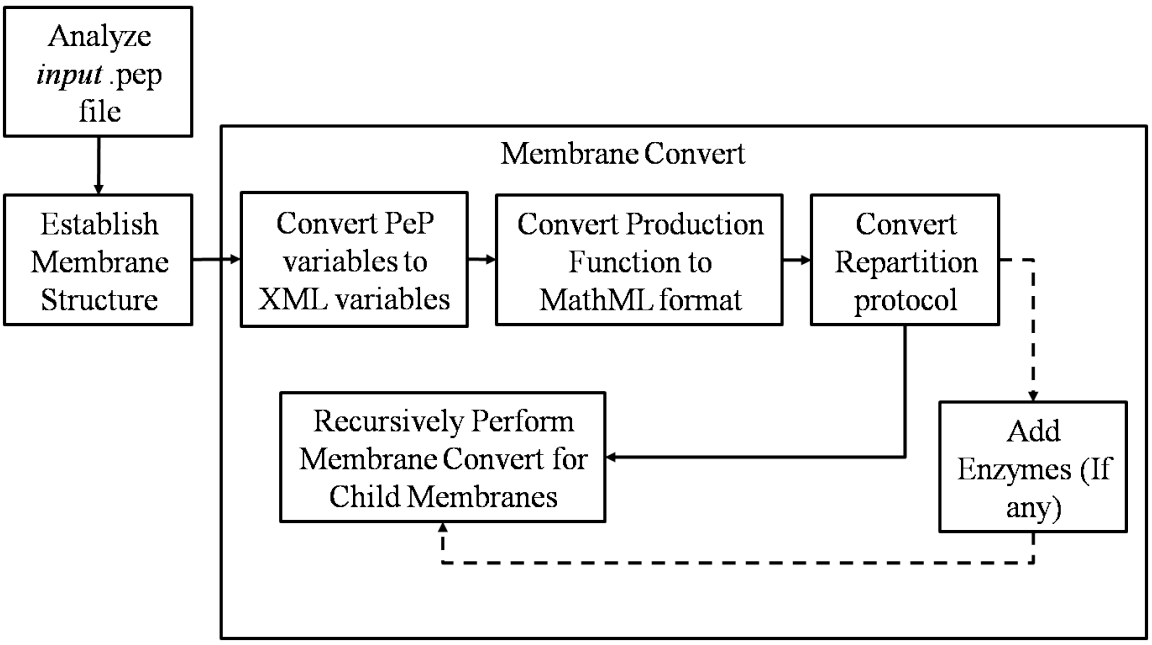

Figure 4.4 File Conversion: PeP to XML

put.xml' is created to store output.

Establish membrane structure:

The 'structure' statement in the file is read and using a stack, the membrane structure is established in the XML file (output.xml) without including internal details of the membranes.

Membrane Convert:

This module is used to convert each individual part of a membrane into its corresponding XML file counterpart (Step 1 in table 4.1). There are several steps for doing it which are elaborated below:

- *Convert variable list to XML:* The variable and variable value list are read and the corresponding variable tags are created in the 'memory' region of the 'output.xml' file. (Step 2 in table 4.1)

- *Convert production functions to MathML:* The production function part of each of the programs present in the membrane is read and converted into MathML, Mathematical Markup Language, an application of XML for describing mathematical notations and capturing both its structure and content (Step 3 in table 4.1). Each program is stored under a rule tag.

- *Convert the re-partition protocol:* The re-partition protocol is converted to a suitable form to insert re-partition variable tags in the output file with the respective contribution attribute which describes the proportion of the distribution of the production assigned to the variable (Step 4 in table 4.1).

- *Add enzyme (if any):* If an enzyme is present as a part of any production function, the representative tags are added to the respective rule in the output file. (Step 5 in table 4.1)

- *Recursively perform MembraneConvert to children membranes:* The child membranes of the membrane considered are again given to the MembraneConvert module to reflect respective changes and additions in the output file. The final output is

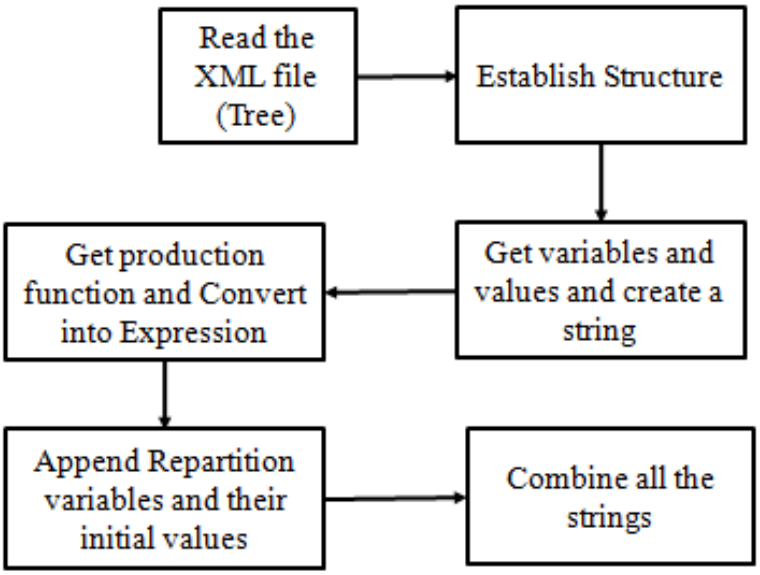

Figure 4.5 File Conversion: XML to PeP

in 'output.xml'

4.1.4 Conversion of files from XML to PeP format

In figure 4.5 the process of conversion from XML to PeP file is presented with the individual components explained as follows:

The file can be run by using

Python3 xmltopep.py filename.xml

Read the XML file (Tree): First the whole XML tree is read tag by tag using 'Element tree' Python library iterating over all the elements. The root of the element is obtained after parsing the whole document and *getiterator* method is called on the root of the tree. This is to make sure that all the elements are read in the file.

Establish Structure: Get 'membranes', 'regions' and 'children' tag are the elements that are stored sequentially as they are read . This is mainly to know the structure of the XML document (Step 1 in table 4.1). The tags are stored in a stack, and are popped out one by one; based on sequence of these elements the structure of the whole membrane is created.

Get variable names and their initial values: This includes both the variables used in the production functions along with the enzymes and their values, if any. (Step 2 in table 4.1)

Get Production function and Convert into Expression: The MathML part which forms the production function is read and converted into a proper expression (notation) (Step 3 in table 4.1).

Append Re-partition variables and their initial values: Repartition variables and their contribution values are appended to the production functions. The variable names are obtained from the tag 'text' and from the attributes their contribution values are known, which are finally appended to the production functions after making a string out of them. (Step 4 and 5 in table 4.1)

Combine all the strings: The strings obtained in the different steps above are merged accordingly and are put into a single file.

Transfer of Values between files of format PeP and execution

An essential feature of Membrane computing is the interaction between membranes in membrane systems. Similar to this the developed tool helps in bring is communication between two PeP files, i.e., not just membranes within a membrane system, but between two different membrane systems. This feature of the tool works on two input PeP files. The values (of some selected variables) obtained on the execution of one of the PeP files is stored and transferred to the other PeP file. The sample output of PeP simulator is shown in figure 4.7. This file is then executed for a certain number of cycles and the final output is displayed.

There are several modules which have been explained as follows: Read and display files in current directory:
The file can be run using the command

Python3 transferValuesPep.py

After upon opening, the current directory is searched for the files of the format PeP

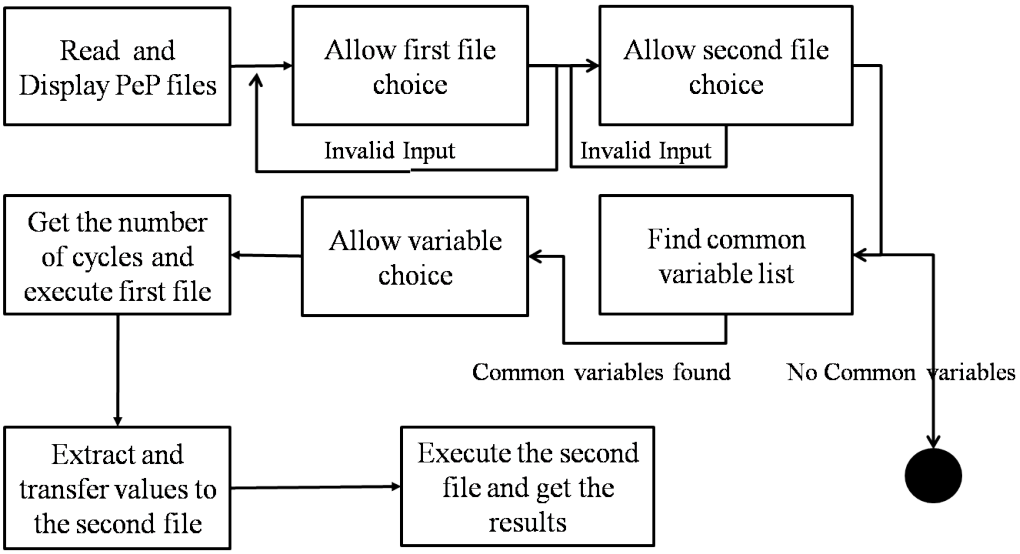

Figure 4.6 Value Transfer: to PeP

```
num_ps = {
  m1:
    var = { x_1_1: 0.21,  x_2_1: 0.02,  x_3_1
: 0.04,  y_1_1: 0.07,  y_2_1: 0.02,  y_3_1: 0
.01,  z_1_1: 0.14,  z_2_1: 0.01,  z_3_1: 0.03
, }
    E = { e_1_1: 40.00, }
  m2:
    var = { x_1_2: 0.07,  x_2_2: 0.01,  x_3_2
: 0.01,  x_4_2: 0.11,  x_5_2: 0.46,  x_6_2: 0
.53,  x_7_2: 0.00, }
    E = { e_1_2: 82.00, }
}
```

Figure 4.7 Output format PeP

and a list of files to choose from is displayed. Each file has a number next to it to help in selection of a specific file.

Allow first and second file choice: The first and second file choices are requested and correspond to the respective number associated with the required file in the aforementioned numbered list. Invalid inputs, such as character or string inputs, as well as numbers not corresponding to any of the files displayed are alerted and the user is asked to re-enter his/her choice. Upon entering valid inputs, the names of the files chosen are displayed.

Find common variables list: Upon obtaining two valid inputs for file choices, the script reads the two PeP files and establishes a list of variables common to both files. If the two files have no variables in common, a relevant error message is displayed and the execution of the script is aborted. Upon obtaining a non-empty set of common variables, the execution is continued.

Allow variable choice: A numbered list of common variables between both the files is displayed and the user is asked to enter his choices. The choices correspond to the number associated with each variable. If a variable already chosen is chosen again, a relevant alert is displayed. An invalid choice such as a character or a string is alerted and the user is pushed to enter a choice again. The final list is decided as the user enters a number greater than the number of common variables between the files. The list of common variables selected is displayed.

Get number of cycles and execute first file: The number of cycles (a positive integer) is requested to be input. If the input format not proper, the user is prompted to enter it again. Upon receiving integer input, the condition that it is a valid number of cycles is checked, and if not, the user is prompted to enter a new number of cycles. The phrase 'valid count' is displayed on receiving an integer representing a valid number of cycles. The 'pep.py' simulator file is used for the execution of the files.

Extract and transfer values to the second file: After successful execution of the first PeP file, the output obtained is parsed and the output values for the required chosen variables are stored. Then a copy of the second file is created and these values are transferred to the

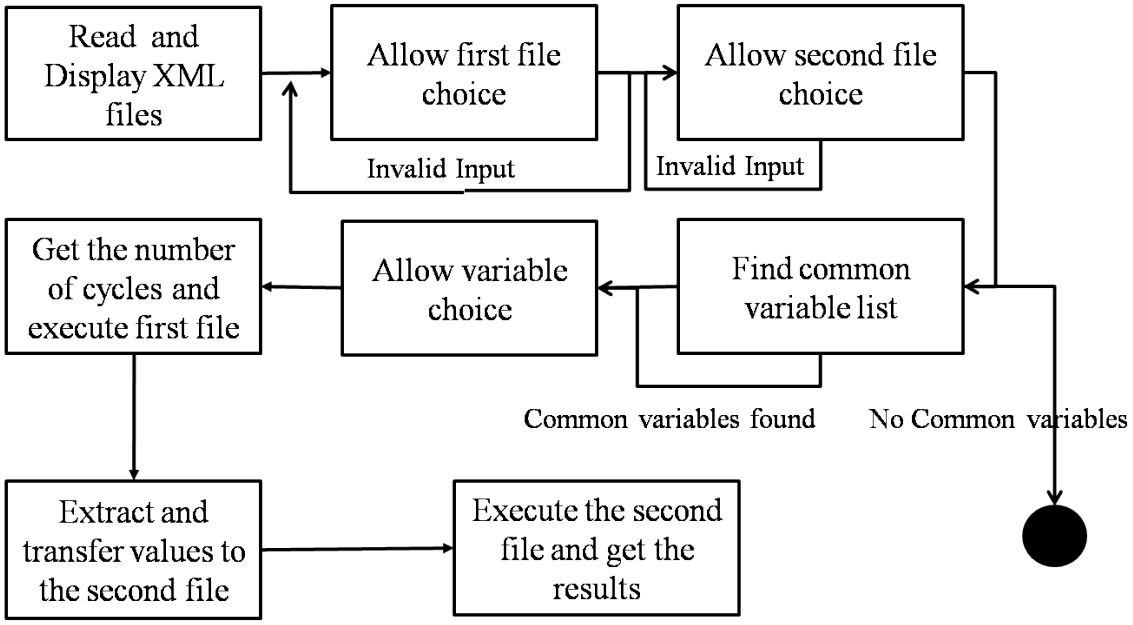

Figure 4.8 Value Transfer: XML to XML

new PeP file (the copy).

Execute the second file and output final result: The number of cycles (a positive integer) is requested to be input. If the input is not of the proper format, the user is prompted to enter it again. The new file with transferred variable values is used as an input to the PeP simulator file. Upon receiving integer inpu1t, the condition that it has a valid number of cycles is checked, and if not, the user is prompted to enter a new number of cycles. The phrase 'valid count' is displayed on receiving an integer representing a valid number of cycles. The same 'pep.py' file is used for the execution. Upon successful execution, the final output is displayed.

Transfer of Values between XML files and execution

The process of transferring variable values between two XML files is very similar to the process of its PeP file counterpart, the only difference being the usage of the *ENPSJava*.jar file for the simulation of the file. The steps involved in the process are as in the figure 4.8. The sample output file of ENPSJava (García Quismondo et al., 2012a) (based on XML file) is shown in figure 4.9. The following file can be executed for this module.

Python3 transferValuesXml.py

```
    Outputs: {z_3_1:0.0,0.0,false, x_7_2:0.0,0.0,false,
 x_6_2:0.0,0.0,false, x_5_2:0.0,0.0,false, x_4_2:3.7473
380988537404,0.0,false, y_1_1:0.0,0.0,false, y_3_1:0.0,
0.0,false, y_2_1:0.0,0.0,false, x_3_1:0.0,0.0,false, x_
2_2:1.7008268430839244,0.0,false, x_3_2:0.34568441268858
914,0.0,false, x_1_1:0.0,0.0,false, x_2_1:0.0,0.0,false
, x_1_2:1.7008268430839244,0.0,false, z_2_1:0.0,0.0,fal
se, z_1_1:0.0,0.0,false}
```

Figure 4.9 Output format XML

Limitations and Dependencies

- The PeP to XML converter can convert PeP files with each production function having less than or equal to a 100 variables.
- The software dependencies required for the tool to work are:
  - Python 3.0
  - The ElementTree XML API
  - Pep.py simulator file and jar version of ENPSJava simulator file in the working directory of the transferValuesPep and the transferValuesXml files respectively.
  - Sympy (Symbolic Python) library

4.1.5 Usecases and correctness of the tool

Achieving Function goals and Design goals

The functional goals and design goals that have been proposed are achieved by the tool. These are elaborated below:

Functional Goals

Conversion of PeP to XML files: The tool does a proper conversion of the files, in which human intervention is required.

Conversion of XML to PeP files: The tool does an errorless conversion of files from XML

to PeP format without human intervention, independent of any external parameters.

Value transfer between two PeP files after execution: Errorless, fault-tolerant transfer of values between two PeP files and seamless execution and results are displayed.

Value transfer between two XML files after execution: Errorless, fault-tolerant transfer of values between two XML files and seamless execution and results are displayed.

Design Goals

Clear Programming Model: Python is used as the base and the tool is developed using a modular approach, which is easy to understand.

Usability: Command line interface is used as it is found to be best for a limited user interaction. The user gets clear instructions wherever required.

Usage of existing or easily available open source software: As mentioned earlier Python is used for creating the tool. The other two P System simulators ENPSJava (Garcıa-Quismondo, 2013) and PeP (Florea and Buiu, 2018) are both opensource.

Scalability: The tool is scalable to bigger files. Case studies shows that the file conversion and multiple execution can be performed for membranes even with more than 10000 programs.

Robustness: There are several checks provided in the tool to ensure that bad data, improper options and several other unintended mistakes by the user are considered and handled properly. In most of the cases the user is notified about the error and is allowed to re-enter it (in the immediate previous step) without as in moving again control back to the first step.

Usecases for the tool

There are primarily three usecases of the tool.

Converting XML files into PeP files

The conversion of XML file into PeP files is useful and has many applications. In this case several legacy files that have been used for several types of application are XML files,

given as input and these have been primarily used with ENPSJava and other similar tools which supports that file system. Some works, like the XML file representing the robot localization problem, which have never been represented in the PeP format before, can now be converted to PeP format, improving readability and further can be executed in PeP simulator for further analysis. One of the significant advantage of this converter is that it allows converting the old, important membrane structures (XML format) into the latest file format and allows it to be used with the newer simulator. Further the XML based files are comparatively bigger than the PeP counterpart. As the number of programs increase the file size, XML files also increases tremendously, whereas, there is a minimal increase in the file size for PeP. Hence, converting the files and using PeP simulator is advantageous.

Converting PeP files into XML files:

PeP file can be converted to XML to execute in the Java based simulator. This allows to test newer membrane systems defined using PeP based format in Java based simulator, while, allowing to test the efficiency of the proposed method in multiple simulators, without any extra effort. PeP files are readable and closely relate to the actual structure of membrane hence even when a membrane structure has to be tested using Java simulator, this converter can be used for designing membrane structure using PeP and then executing in Java based simulator.

Interaction between Multiple Membranes

This tool allows connecting membrane systems without manually copying the data from membrane system into the other membrane system, after execution. This further allows using a chain of membrane systems with serial dependence to be connected by passing data between them, in order, without any manual copying and automatic execution of all the membrane systems in the same order. This is designed for both the file formats and hence can be used for both the simulators. An important use case is for solving bigger problems which involve different steps (hence designed as different membrane systems). These membrane systems can be connected together to get the final output.

Checking the correctness of the converters

The converters are designed such that it takes care of a few semantic discrepancies on the side of the user, specifically in the case of giving PeP file as input. As the PeP simulator supports, the converter also takes care of extra spaces in starting and end of line, extra single spaces over the braces, variables and enzymes. Thus small mistakes from the users side cannot effect the converter.

Once a PeP file is converted into XML file, the converted file is reconverted back into PeP. The structure of the converted file is checked with the original file. If it is same then the converter works correctly. Also the files can be executed using Java ENPS simulator. If the reconverted file gives the same results as supposedly given by the original PeP file then both the converters are proper. Similarly this can be checked in another way where a given XML file is converted into a PeP file and then converted back to XML file. This XML file is tested and the results, are matched with the original files results for its correctness. These tests have been followed in case studies. This ascertains the functional correctness of the converter.

4.1.6 Case Studies

This section considers different scenarios and cases for the tool that demonstrate the correctness of the tool. This has been divided into three subsection which have been elaborated as follows:

Case Study 1: Automatic Transfer of values (XML to XML and PeP to PeP)

To demonstrate the value transfer (data passing) property of the tool, a set of functions is considered. Each function can perform a set of tasks, in this case it is defined as a mathematical equation. Consider any mathematical function which can perform certain set of operations. Each function can be considered as a membrane system (consisting of one or more membranes), which can perform certain operation that is feasible by P System. The tool allows the membranes (functions) to pass the values to other membranes (functions). The tool provides the user with the list of values that can be transferred and the

user can select the values to pass. The tool automatically passes the values after execution of the first and before the execution of the second membrane.

Consider the following simple equation:

$$L(x,y,z) = (a^x + b^x + c^x + d^x)/e^y * f^z \tag{4.1}$$

where, a, b, c, d, e and f are constants and x, y and z are variables

The whole expression is divided into three functions, F(x), G(y) and H(z) and the equation is written as in equation 4.5. The membrane structure of each function is as follows:

$$F(x) = (a^x + b^x + c^x + d^x) \tag{4.2}$$

$$G(F(x), y) = F(x)/e^y \tag{4.3}$$

$$H(G(F(x), y), z) = G(y) * f^z \tag{4.4}$$

The equation is written as:

$$L(x,y,z) = H(G(F(x), y), z) \tag{4.5}$$

Each and every function is converted into membrane structure.

The equivalent membrane structure for $F(x)$ is as in figure 4.10, $G(y)$ is shown figure 4.11 and $H(z)$ is figure 4.12.

$\underline{F(x) - M_1}$

$$x_{1,1}[0], x_{2,1}[b], x_{3,1}[c], x_{4,1}[d], x_{5,1}[a], x_{6,1}[b], x_{7,1}[c], x_{8,1}[d], x_{9,1}[a]$$

$$e_{1,1}[200000]$$

$$Pr_{1,1}: \; x_{5,1} + x_{1,1} - x1, 1 \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{5,1}$$

$$Pr_{2,1}: \;\; 2 * x_{9,1} * x_{5,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{9,1} + 1 \mid x_{1,1}$$

$$Pr_{3,1}: \; x_{6,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{6,1}$$

$$Pr_{4,1}: \;\; 2 * x_{2,1} * x_{6,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{2,1} + 1 \mid x_{1,1}$$

$$Pr_{5,1}: \; x_{7,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{7,1}$$

$$Pr_{6,1}: \;\; 2 * x_{3,1} * x_{7,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{3,1} + 1 \mid x_{1,1}$$

$$Pr_{7,1}: \; x_{8,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{8,1}$$

$$Pr_{8,1}: \;\; 2 * x_{4,1} * x_{8,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{4,1} + 1 \mid x_{1,1}$$

Figure 4.10 Membrane System for Function 1

$\underline{G(y) - M_2}$

$$x_{1,1}[value], x_{2,1}[e], e_{1,1}[200000], e_{2,1}[0]$$

$$Pr_{1,1}: \; x_{1,1}/x_{2,1} \;\; \langle e_{2,1} \rightarrow \rangle \;\; 1 \mid x_{1,1}$$

$$Pr_{2,1}: \; x_{2,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{2,1}$$

Figure 4.11 Membrane System for Function 2

$\underline{H(z) - M_3}$

$$x_{1,1}[value], x_{2,1}[f], e_{1,1}[200000]$$

$$Pr_{5,1}: \;\; x_{1,1} * x_{2,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{1,1}$$

$$Pr_{6,1}: \; x_{2,1} \;\; \langle e_{1,1} \rightarrow \rangle \;\; 1 \mid x_{2,1}$$

Figure 4.12 Membrane System for Function 3

This shows a simple case where a single value is passed from one membrane to another membrane structure. As mentioned, three membranes are considered, each membrane has an operation (function) to be performed. The first membrane calculates the final value of equation 4.2 using the corresponding membrane as depicted in figure 4.10. The final value from a set of variables of membrane 1 (figure 4.10) is passed. Similarly a value from set

of variables of membrane 2 (figure 4.11), evaluating equation 4.3 is passed to membrane 3 (figure 4.12) which finally calculates equation 4.4. The membrane structures are executed with constant values (a, b, c, d, e and f) given inside the membrane (as variables) and the variables (x, y and z) given as cycles for each membrane respectively.

The next example deals with a set of membranes where more than one value is passed to another membrane. Considering equation 4.6 the membrane details are given in figure 4 and 5. Here two values of $x_{1,1}$ (final series value after addition) and $x_{2,1}$ (the factorial value) are passed. Finally these two values are used in the next (dependent) membrane.

Similar to this, several complex problems also can be solved, where each membrane can compute a set of operations and share the data among themselves. The set of inputs considered for both; variables are passed in PeP file and variables are passed in XML file. Table 4.2 gives the results of all the sample cases tested, using the equation; the actual value and values are obtained after executing in both the simulators with different set of inputs.

$$L(x,y) = (((\sum_{i=1}^{x} n^i)/x!)^{1/y}) \tag{4.6}$$

$\underline{L(x,y) - M_1}$

$$x_{1,1}[0], x_{2,1}[1], x_{3,1}[n], x_{4,1}[1], x_{5,1}[n], e_{1,1}[200000]$$
$$Pr_{1,1} : x_{3,1} * x_{5,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{5,1}$$
$$Pr_{2,1} : x_{5,1} + x_{1,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{1,1}$$
$$Pr_{3,1} : x_{3,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{3,1}$$
$$Pr_{4,1} : x_{4,1} + 1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{4,1}$$
$$Pr_{5,1} : x_{4,1} * x_{2,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{2,1}$$

Figure 4.13 Membrane System for Function 1 - Problem 2

$\underline{L(x,y) - M_2}$

$$x_{1,1}[value1], x_{2,1}[value2], x_{3,1}[0], x_{4,1}[y], e_{1,1}[200000], e_{2,1}[0]$$

$$Pr_{1,1} : \ x_{1,1}/x_{2,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{3,1}$$

$$Pr_{2,1} : \ x_{3,1}^{1/x_{4,1}} \ \langle e_{2,1} \rightarrow \rangle \ 1 \mid x_{1,1}$$

$$Pr_{3,1} : \ e_{1,1} + e_{2,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid e_{2,1}$$

$$Pr_{4,1} : \ e_{2,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid e_{1,1}$$

Figure 4.14 Membrane System for Function 1 - Problem 2

Equation	**Inputs**									**Output (PeP)**	**Output (XML)**	**Actual Value**
$L(x,y,z) = (a^x + b^x + c^x + d^x) / e^y * f^z$	**a**	**b**	**c**	**d**	**e**	**f**	**x**	**y**	**z**	**L(x,y,z)**	**L(x,y,z)**	**L(x,y,z)**
	1	2	3	4	3	4	3	4	5	1264.19	1264.19	1264.19
	12	16	2	5	4	6	4	8	4	1718.74	1718.74	1718.74
	2	4	8	16	4	4	3	3	3	4680	4680	4680
	10	20	32	16	4	8	2	4	4	28480	28480	28480
$L(x,y) = (((\sum_{i=1}^{x} n^i)/x!)^{1/z})$	**n**			**x**			**y**			**L(x,y)**	**L(x,y)**	**L(x,y)**
	2			8			4			0.335361	0.335361	0.335361
	5			9			2			2.593811	2.593811	2.593811
	8			6			5			3.340716	3.340716	3.340716
	12			3			3			6.796884	6.796884	6.796884

Table 4.2 Membranes equivalent values

Case Study 2: Convert PeP to XML

For testing the correctness and capability of the converter, two different set of cases have been considered.

Case 1: Correctness of Converter

To check the correctness of the converter, three membranes, as in figure 4.10, 4.11 and 4.12, are considered separately. These three membranes represent equation 4.2, 4.3 and 4.4, respectively. The final value for all the equations for a given set of constants is known

(can be manually calculated) and the converter works correctly if both the files (PeP and XML) give the same value. Table 4.3 shows the final values of the PeP file and XML file for a set of input values. In all the cases the output is as intended.

Case 2: Large File Conversion

This case study is to check the capability of the system to support larger membranes i.e. whether the converter is able to convert larger or bigger files (many normal and dense programs) efficiently without any error. For checking this, two files were considered. The first file is a simple membrane with nearly 10000 programs. The seed file for the membrane is as shown in figure 5.7. A seed membrane is a simple membrane which gives an abstract structure that can be turned into a bigger membrane by replicating the operation (programs and membranes) performed by the seed membrane. The resultant PeP file is successfully converted to XML file. This file is then executed using the ENPSJava Simulator. The output for both the files are the same. Similarly the next file is a more complex file, with multiple membranes, which has nearly 3000 programs considered for conversion. The tool successfully converts the file into the XML format. This resultant file has the same structure and is executed without any error. The seed membrane is as shown in figure 4.16. The outputs show that the tool is able to convert the PeP and XML file properly.

Further, the given files, for both the case studies, are reconverted into the other format after conversion and is checked again i.e. the resultant XML file is converted back to PeP and then is executed with PeP simulator and as expected it gives the same output for all the cases. This ascertains that both the converters work properly, as any discrepancy in either of the converters can lead to a wrong result or a deviation in the result.

$$x_{1,1}[1], x_{2,1}[1], x_{3,1}[1], x_{4,1}[1], y_{1,1}[1], y_{2,1}[7], y_{3,1}[1], y_{4,1}[7], y_{1,1}[200000]$$

$$Pr_{1,1} : x_{1,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{1,1}$$
$$Pr_{2,1} : x_{2,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{2,1}$$
$$Pr_{3,1} : x_{3,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{3,1}$$
$$Pr_{4,1} : x_{4,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{4,1}$$
$$Pr_{5,1} : y_{1,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid y_{1,1}$$
$$Pr_{6,1} : y_{2,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid y_{2,1}$$
$$Pr_{7,1} : y_{3,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid y_{3,1}$$
$$Pr_{8,1} : y_{4,1}+1 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid y_{4,1}$$

Figure 4.15 Seed Membrane 1 for Case Study 2 and 3

Case Study 3: Convert XML to PeP

This case study tests the capability of the system, to convert the XML based files to PeP files. Again, there are two cases, Normal file conversion and larger file conversion.

Case 1: Correctness of the Converter

In this case, two membranes, as shown in figure 4.10 and figure 4.12, are considered. These figures are equivalent to the corresponding equations 4.2 and 4.4, respectively. These membranes initially in XML format are converted into PeP and are executed with PeP simulator. The results show that the tool converts the files properly as the expected results are obtained (table 4.4).

Further, the resultant files are converted back to XML files. These reconverted files are executed using ENPSJava (Garcıa-Quismondo, 2013) giving the same results for all the cases, ascertaining that both the converters are working properly.

$$x_{1,1}[1], x_{2,1}[1], x_{3,1}[3], x_{4,1}[3], x_{5,1}, [0.6]x_{6,1}[0.7]$$
$$y_{1,1}[0.2], y_{2,1}[0.3], y_{3,1}[0.5], y_{4,1}[5], y_{5,1}[0.8], y_{6,1}[0.9]$$
$$z_{1,1}[3], z_{2,1}[4], z_{3,1}[5], z_{4,1}[3], z_{5,1}[0.5], z_{6,1}[0.7]$$
$$a_{1,1}[2], a_{2,1}[0.5], a_{3,1}[6], a_{4,1}[0.3], a_{5,1}[0.8], a_{6,1}[3]$$
$$b_{1,1}[2], b_{2,1}[3], b_{3,1}[0.7], b_{4,1}[0.9], b_{5,1}[0.8], b_{6,1}[0.9]$$
$$c_{1,1}[3], c_{2,1}[2], c_{3,1}[4], c_{4,1}[0.5], c_{5,1}[7], c_{6,1}[0.9]$$
$$f_{1,1}[0], f_{2,1}[0], f_{3,1}[0], f_{4,1}[0], f_{5,1}[0], f_{6,1}[0], e_{1,1}[200000]$$

$$Pr_{1,1}: 7*(x_{1,1}*x_{2,1}*x_{3,1}/x_{4,1}*x_{5,1}*x_{6,1})^{1/0.166} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid f_{1,1}$$
$$+1|x_{1,1}+1|x_{2,1}+1|x_{3,1}+1|x_{4,1}+1|x_{5,1}+1|x_{6,1}$$
$$Pr_{2,1}: 7*(y_{1,1}*y_{2,1}*y_{3,1}/y_{4,1}*y_{5,1}*y_{6,1})^{1/0.166} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid f_{2,1}$$
$$+1|y_{1,1}+1|y_{2,1}+1|y_{3,1}+1|y_{4,1}+1|y_{5,1}+1|y_{6,1}$$
$$Pr_{2,1}: 7*(z_{1,1}*z_{2,1}*z_{3,1}/z_{4,1}*z_{5,1}*z_{6,1})^{1/0.166} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid f_{3,1}$$
$$+1|z_{1,1}+1|z_{2,1}+1|z_{3,1}+1|z_{4,1}+1|z_{5,1}+1|z_{6,1}$$
$$Pr_{4,1}: 7*(a_{1,1}*a_{2,1}*a_{3,1}/a_{4,1}*a_{5,1}*a_{6,1})^{1/0.166} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid f_{4,1}$$
$$+1|a_{1,1}+1|a_{2,1}+1|a_{3,1}+1|a_{4,1}+1|a_{5,1}+1|a_{6,1}$$
$$Pr_{5,1}: 7*(b_{1,1}*b_{2,1}*b_{3,1}/b_{4,1}*b_{5,1}*b_{6,1})^{1/0.166} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid f_{5,1}$$
$$+1|b_{1,1}+1|b_{2,1}+1|b_{3,1}+1|b_{4,1}+1|b_{5,1}+1|b_{6,1}$$
$$Pr_{6,1}: 7*(c_{1,1}*c_{2,1}*c_{3,1}/c_{4,1}*c_{5,1}*c_{6,1})^{1/0.166} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid f_{6,1}$$
$$+1|c_{1,1}+1|c_{2,1}+1|c_{3,1}+1|c_{4,1}+1|c_{5,1}+1|c_{6,1}$$
$$Pr_{7,1}: x_{3,1}*0.0015 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid x_{3,1}$$
$$Pr_{8,1}: y_{3,1}*0.0015 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid y_{3,1}$$
$$Pr_{9,1}: z_{3,1}*0.0015 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid z_{3,1}$$
$$Pr_{10,1}: a_{3,1}*0.0015 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid a_{3,1}$$
$$Pr_{11,1}: b_{4,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid b_{4,1}$$
$$Pr_{12,1}: c_{3,1}*0.0015 \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid c_{3,1}$$

Figure 4.16 Seed Membrane 2 for Case study 2 and 3 (Complex Programs)

Case 2: Large file conversion

Here the capability of the system to execute large files is tested. In this case, two types of files are considered; a file (membrane) consisting of simple programs, upto 10000 in numbers (multiple membranes). Another one is a membrane system consisting of 3000

complex programs with multiple membranes. Complex programs are given, so as to check the capability of the converter to convert dense programs. The number of programs considered in this study are just enough to demonstrate large files, the tool based on its structure and structure that can support much more number of programs without possible errors. Both of these cases tend to work as intended.

Equation	**Inputs**					**Output (PeP)**	**Output (XML)**
$F(x) = (a^x + b^x + c^x + d^x)$	**a**	**b**	**c**	**d**	**x**	**F(x)**	**F(x)**
	1	2	3	4	3	100	100
	12	16	2	5	6	19778889	19778889
	2	4	8	16	4	69904	69904
	10	20	32	16	5	37903008	37903008
$H(z) = n * f^z$	**n**		**f**		**z**	**H(z)**	**H(z)**
	5		3		4	405	405
	10		4		8	655360	655360
	7		8		5	229376	229376
	16		5		7	1250000	1250000

Table 4.3 Results for Case study 2

4.2 GPUPeP

ENPS is a class of P System in which membranes operate on numerical values. To realize the power of ENPS there are a few simulators developed. Each and every simulator has some advantages as well as some disadvantages. For this research a GPU based simulator using Python as a user interaction language is developed. This tool is a completely parallel variant, compatible with a Python-based sequential simulator (PeP), which is the first Python based work for ENPS. The developed simulator uses CUDA to interact with GPU and it gives the desired speed up, while processing the membranes. There are two important case studies which show the performance of the developed tool to be far better than the other serial simulators.

Equation	Inputs					Output (XML)	Output (PeP)
$F(x) = (a^x + b^x + c^x + d^x)$	**a**	**b**	**c**	**d**	**x**	**F(x)**	**F(x)**
	2	1	4	3	5	1300	1300
	5	2	12	8	4	25473	25473
	4	8	2	4	8	16908544	16908544
	25	4	32	16	5	44369657	44369657
$H(z) = n * f^z$	**n**		**f**		**z**	**H(z)**	**H(z)**
	2		5		6	31250	31250
	5		6		7	1399680	1399680
	8		3		12	4251528	4251528
	3		4		5	3072	3072

Table 4.4 Results for Case study 3

4.2.1 Introduction

The concept of membrane computing has been introduced by Gheorghe Paun (Dassow and Păun, 1999) and further details has been given in his seminal paper in 2000 (Păun, 2000). Membrane computing, a branch of natural computing, is a computing paradigm inspired by the structure of a living cell. P Systems are the computing devices that are used to realize membrane computing models. There are several types of P System models designed, based on the structure of the cells. There are primarily three structural models, Tissue-like P System, Cell-like P system and Neuron based P System. This study deals with cell-like P Systems. This membrane model consists of multiple membranes placed in a hierarchical structure, with a single membrane collectively bounding all the membranes. Any number of membranes can be placed inside the cell membrane, with the property of possible communication between one another i.e. they can pass information to each other. The P System structure is an inherently parallel structure, and this property gives an undue advantage to the model over other models. It is a computing paradigm, unlike other similar nature-inspired models, which are usually methods and approaches for specific purposes (Martínez-Puras and Pacheco, 2016; Mohammadi et al., 2018; Eusuff et al., 2006). P

System is one of the widely used unconventional computing models.

P System has several variants, with every variant having its own application and properties, except parallelism and multi-membrane structure, which is common for all the variants of cell-like P System. This work concentrates on the Numerical P System (NPS), a variant of P System. A variant of the Numerical P systems framework called Enzymatic Numerical P systems (ENPS) is considered here. ENPS is Numerical based model that has several applications. To realize its power, several simulation tools have been developed (García Quismondo et al., 2012a; Martínez-del Amor et al., 2015; Florea and Buiu, 2017). Similarly, this work aims at developing a GPU based Python simulator (GPUPeP)[1] for ENPS. This is the second work in Python (as interface) for ENPS after Florea et al. (Florea and Buiu, 2018), who designed a sequential Python tool for ENPS and the same membrane structure is used as input, so as to maintain compatibility between serial and parallel tools.

4.2.2 Design Goals

Following is a list of design goals for the GPUPeP simulator:

Clear Programming Model: A clear programming model refers to the use of a programming model with a proper definition and standard structure accepted by the programming community around the world. This primarily deals with the simulator's programming model, which must be clear, usable, and extensible.

The developed simulator uses Python to get the values from the file. The same procedure by Florea and Buiu (2018) is followed in this study for reading and storing the input (objects), for the initial phase. The input file format used here is the same as that given by Florea et al. (Florea and Buiu, 2017, 2018), as it closely resembles the formal structure of ENPS System. These features have been retained to have compatibility between these two tools, which can later be combined as a single entity for sequential and parallel execution if required. The CUDA simulator uses the Python object representing the ENPS model for parallel simulation for execution. CUDA is again a well-defined framework and one of the

[1]The tool is available on request from the authors

efficient ways to use NVIDIA GPUs (Nvidia, 2019).

Performance: The GPUPeP simulator's core is written in CUDA C. CUDA is NVIDIA's parallel computing platform and an application programming interface, that gives a significant increase in performance by enabling the programmer to harness the power of GPUs. The case studies show that for ENPS models, with a large number of programs (approx. 10000) and a large number of steps (approx. 10000), the speed-up obtained is nearly 770x compared to the sequential simulator.

Ease of Understanding: The user interfaces part of the simulator is written in Python (specifically, for reading and storing the membrane structure). Every ENPS model is converted to Python object, which can easily be operated upon (Florea and Buiu, 2018). GPUPeP uses these Python objects and passes the required data to CUDA C and runs the parallel simulation.

Simple Interface: The simulator is running on a terminal. The input to the simulator is a *pep* file, which is a simple representation of the ENPS model (Florea and Buiu, 2018). The input flags given by the user sets the output of the simulator.

Single Input Format: The GPUPeP simulator expects the ENPS model in a certain format as input. The Python simulator converts the *pep* file into Python object, which is then converted into the relevant format as required by the CUDA C program.

4.2.3 Design and Implementation of the Tool

The research community has indicated the requirement for an easy to use parallel ENPS simulator. The existing parallel simulators are either based on C or Java. Most of them require their input to be in XML format. Thus, a decision to develop upon the Python-based simulator for the ENPS, that takes input in *pep* format (which replicates the exact structure of a membrane, making it more appealing to the users) has been taken. This model of input has been proposed and used first by Florea and Buiu (2018, 2017), which is an easily comprehensible object-based structure. The file extension '.pep' is used to refer to input files for simulator in the presented study also. This is considered as a standard

```
num_ps = {
      # membrane names (labels)
      H = {m1, m2};

      structure = [m1 [m2 ]m2 ]m1;

      # membrane 1
      m1 = {
      var = {x_1_1, x_2_1, x_3_1}; # variables used in the production function
      E = {e_1_1, e_2_1}; # set of enzyme variables
      pr = {2*x_1_1 + x_2_1 [e_1_1 -> ] 1|x_2_1 + 1|x_3_1 + 1|x_1_2};
      pr = {x_1_1 + 4*x_3_1 [e_2_1 -> ] 1|x_1_1 + 2|x_2_1};
      var0 = (2, 3, 4); # initial values for variables x_1_1, x_2_1, x_3_1
      E0  = (4, 1); # initial values for enzymes e_1_1, e_2_1
      };

      m2 = {
      var = {x_1_2, x_2_2}; # variables used in the production function
      E = {e_1_2}; # set of enzyme variables
      pr = {x_1_2 + x_2_2 [e_1_2 -> ] 1|x_1_2 + 1|x_2_2 + 1|x_2_1};
      var0 = (3, 2);
      E0  = (5);
      };
}
```

Figure 4.17 PeP file format (Florea and Buiu, 2017, 2018)

format for input. Though a similar user-friendly format may have been created for this tool, it is found that creating a new one would again restrict the interoperability of the tool. Using *pep* allows the users to use both the tools interchangeably for testing. Figure 4.17 shows a sample structure of the input file (Florea and Buiu, 2018, 2017):

A brief explanation of the input format:

- **H:** contains the list of membranes in the ENPS
- **structure:** stores info on how the membranes are present in ENPS. Suppose there were two separate membranes m1 and m2, it would be [m0 [m1]m1 [m2]m2]m0 where, m0 is the skin membrane under which the individual membranes are contained and if only m1 was inside m0, it is [m0 [m1]m1]m0. Note that one membrane cannot be both inside and outside another membrane
- **m1** (1 here refers to the first membrane, which is according to general naming convention (mx)): contains all the information regarding that membrane. It is the wrapper of a number of data structures such as the variables, programs, enzymes, etc.

which belong to that membrane.

- **var:** contains a list of the variables that belong to the membrane. The variables are named in the form x_a_b where 'x' represents the variable name, 'a' represents the membrane number and 'b' represents the variable number in that membrane. This is a convention, the variable name can be any string.

- **E:** contains a list of enzyme variables belonging to that membrane in the same format as that of the variables

- **pr:** represents a program that belongs to the membrane. It usually consists of a production function, an enzymatic check, and a distribution function along with their proportions in a mathematical format

- **var0:** contains the values of the variables belonging to the same membrane in the same order as mentioned in 'var'

- **E0:** contains the values of the enzyme variables belonging to the same membrane and in the same order as mentioned in 'E'

Python's Regex package is used to convert into Pyhon Classes, the input in the specified format in the '.pep' files. Separate functions are defined to convert the parsed infix production function into its corresponding postfix expressions, which are then evaluated with the given values and distributed among the variables in the distribution functions. Distribution rules are followed while doing so.

Once the *pep* file is read and stored in the proper format, the control moves to CUDA program, which is the primary contribution of this work. As mentioned, the Python part of the simulator parses the *pep* file and converts the given ENPS model into Python objects (Florea and Buiu, 2017). The CUDA simulator is written in CUDA C. The Python objects are parsed into an appropriate format and given as input to the CUDA simulator. The transfer of information from the Python simulator to CUDA simulator is via a file. The PeP parser (Florea and Buiu, 2018) has been reused in this study for reading the *pep* file primarily because of its effectiveness in perfectly reading the *pep* structure perfectly.

CUDA Programming

A brief description of CUDA programming is given (Cook, 2012). In CUDA, the device refers to GPU, while the host refers to CPU. The host and device have a master-slave relationship. The host can manage memory in host and device and make calls to the kernel. Many GPU threads execute these kernels in parallel. The main program is executed in the host and thus, the user input is transferred from host to the device by the host.

Given the heterogeneous nature of the CUDA programming model (Cook, 2012), a typical sequence of operations for a CUDA C program is:

1. Initialization of host data.
2. Transfer of data from the host to the device.
3. Execute one or more kernels.
4. Transfer computed results back from host to device.
5. De-allocation of allocated memory in host and device.

All of these steps have been followed in the present work

The computations done by the CUDA simulator can be broadly classified into the following three stages:

- Evaluate the program's production function, with the values of the membrane, at the start of that step.
- Evaluate the variables in the production function, against the enzyme variable, for that program and proceed if all the variables are less than the enzyme.
- Calculate the distribution fraction for each variable in the distribution function, using the distribution proportions and distribute the produced value accordingly.

A CUDA kernel is defined in the CUDA simulator for parallel simulation of the ENPS model. The computations done inside the CUDA kernel is as follows:

- Parse production function into postfix expression:
  - Production functions of all programs are parsed and converted into a postfix expression.
  - This is a one-time process done for faster evaluation of all production functions in each step.
- Parse distribution function into an operable data structure
  - Distribution function passed as a string is parsed and converted into an operable data structure (C struct).
  - The proportions in the repartition protocol are normalized.
  - Normalisation is not necessary but is done to avoid recalculating the amount to be distributed amongst variables in the distribution function.
- Simulate the membrane
  - Simulate the programs as per the number of cycles given
  - Calculate the production function of each program.
  - For each program, set its activation. If the program is inactive, then ignore the result of the computation is ignored in its production function.
  - Set values of consumed variables (variables in active production functions) to zero.
  - For active programs, distribute the values of the production function to variables in distribution function according to repartition protocol. The previously mentioned data structure is used for this. This step is done atomically in CUDA.

The kernel spawns a single block. Each program is mapped to a thread. If there are more than 1024 programs, they are evenly distributed among threads. The entire simulation of the ENPS programs happens within a single kernel. This approach has been chosen

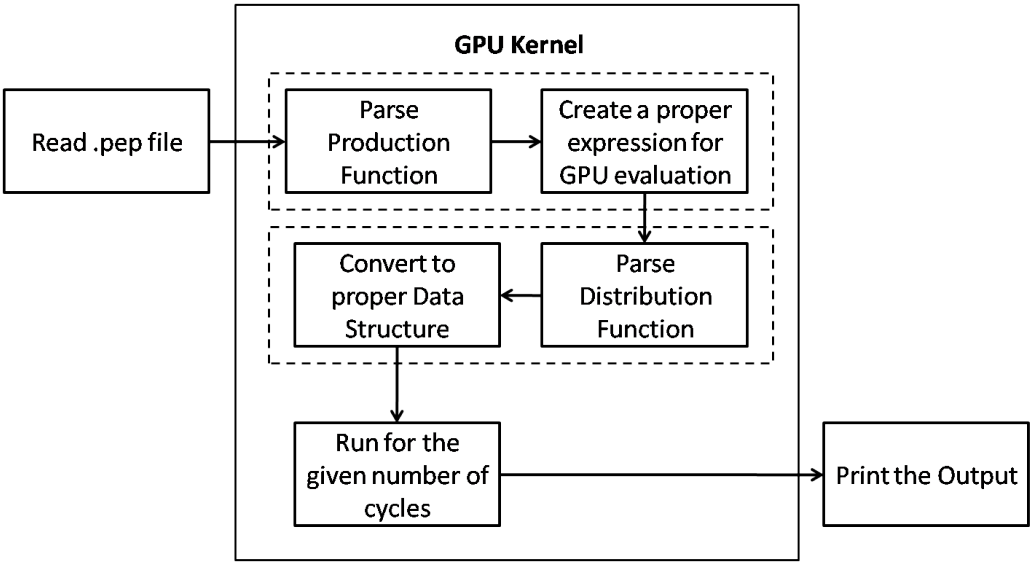

Figure 4.18 GPU Kernel level Operations

over launching several small kernels that do specific parts of the simulation. The latter approach may have given significant kernel launch overhead (when summed across each kernel launch).

The data layout consists of 11 components namely, following is a brief explanation of data in each line:

- Number of programs
- Number of membranes
- Production function
- Delimiters for production function
- Distribution function
- Delimiters for distribution function
- Values of variables
- Delimiters for variables
- Values of enzymes in each program

- Number of cycles of simulation

- Names of variables corresponding to membrane names

As mentioned, there is another tool (García Quismondo et al., 2012a), which is one of the first tools for ENPS using GPU's. There are two primary differences between the current tool and tool by García Quismondo et al. (2012a), when CUDA architecture is compared. García Quismondo et al. (2012a) uses an expression tree to evaluate production function, whereas this study converts it to postfix expression and evaluate. Unlike the single kernel, several small kernels are defined to handle a specific part of the simulation.

Apart from these differences related to CUDA design, the parser design is in Python, and data layout is different as opposed to C/C++ in (García Quismondo et al., 2012a).

4.2.4 Interaction with Simulator

The simulator uses the same command and membrane structure as given by Florea et al. (Florea and Buiu, 2018). (To maintain compatibility with this Python tool).

The parallel simulator is run by invoking special command as mentioned below (the same file works in the sequential mode as in the above-mentioned tool (Florea and Buiu, 2018), without the parallel option mentioned below):

```
$ ./gpupep.py <PEP_INPUT_FILE> [options]
```

- -n NR: stop the simulation after NR execution steps

- --parallel: switch to parallel execution mode

Running either of the following commands displays the set of options mentioned above (and more):

```
$ ./gpupep.py --{help}
$ ./gpupep.py
```

All user interaction with the simulator can be represented with an interaction diagram (Lopes et al., 2018) (See Figure 4.19).

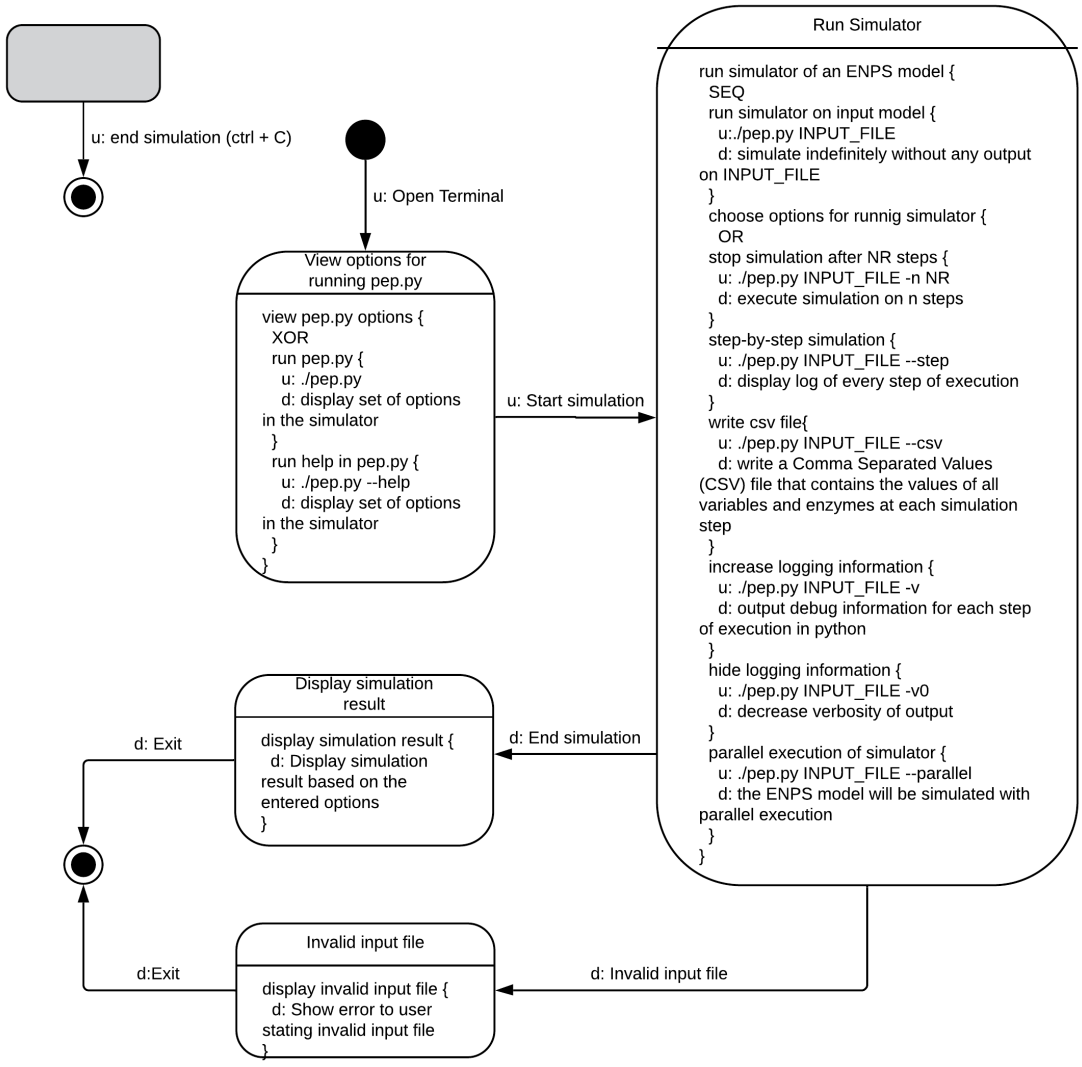

Figure 4.19 Interaction diagram

Following is a brief description of various elements in the diagram:

1. **Opening point:** A filled black circle represents the Opening Point. This indicates where the user starts interacting with the simulator.

2. **Scene:** A rounded rectangle represents a scene. In a scene, the user decides how the conversation flows between her and the system. The top section contains the topic of the scene and represents the user's goal.

3. **Signs:** represent the information involved in the utterances issued by the user (i.e., user input) and by the designer's deputy (i.e., system output) during the dialogues. 'u' for the user and 'd' for the designer. For example $./gpupep.py$ in "Viewing options for running $gpupep.py$ scene".

4. **Dialogues:** It indicates a conversation about a topic. For example, "display invalid input file" dialogue in "Invalid input file" scene.

5. **Structures of dialogue:** The dialogues can be a composition of dialogues. In these cases, the dialogues are combined by operators like SEQ, XOR, OR or AND.

    - Dialogues within SEQ operator must be exchanged in the same sequence.
    - Dialogues within XOR operator are mutually exclusive to one another.
    - Dialogues within OR operator can be exchanges amongst themselves.
    - Dialogues within AND operator must all be exchanged, but not necessarily in the same sequence.

6. **Transition utterance:** Represents the user or the system giving up their turn of interaction for the other. It is represented by an arrow in the diagram, labeled with a user utterance indicator (u:) or designer utterance indicator (d:). It is represented by an arrow labeled by u: (user utterance) or d: (designer utterance).

7. **Ubiquitous access:** Indicates that the user can change the topic of conversation at any point during the simulation. It is represented through a gray rounded rectangle.

For example, the user can press “Ctrl + C”, to terminate the program anytime.

8. **Closing point:** Indicates an end of interaction. A filled black circle within a circle with no padding represents the Closing Point in Figure 2.

4.2.5 Case studies: Testing the Tool

The performance of the tool under different conditions is analyzed in this section. The tool harnesses GPUs’ inherent capabilities and is based on CUDA, thus the tool’s performance depends on these technologies. The improvement in GPUs over time will directly affect the performance of the tool. The use of GPU and an appropriate design of the tool ensures that the tool performs much better than its sequential counterparts. The performance gain is several times more than the other sequential tools present. This section is further divided into two, where the tool’s performance for different test data is analyzed.

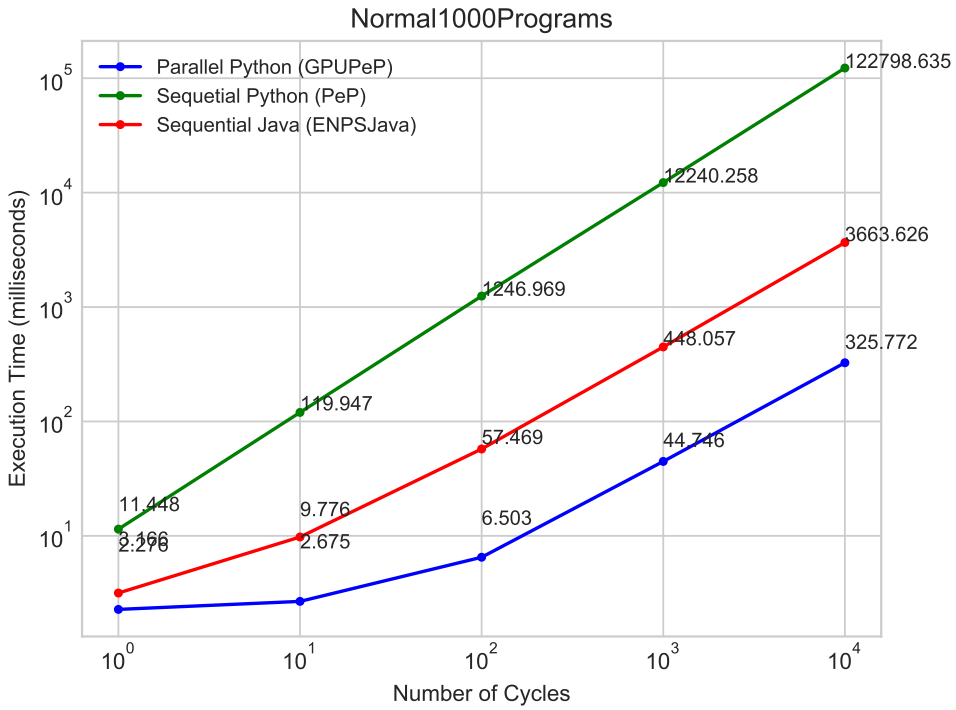

Figure 4.20 Normal 1000 Programs

GPUPeP comes in two variants, float and double. In float variant, the final variable values are in a single precision floating point whereas, in double variant, they are in double

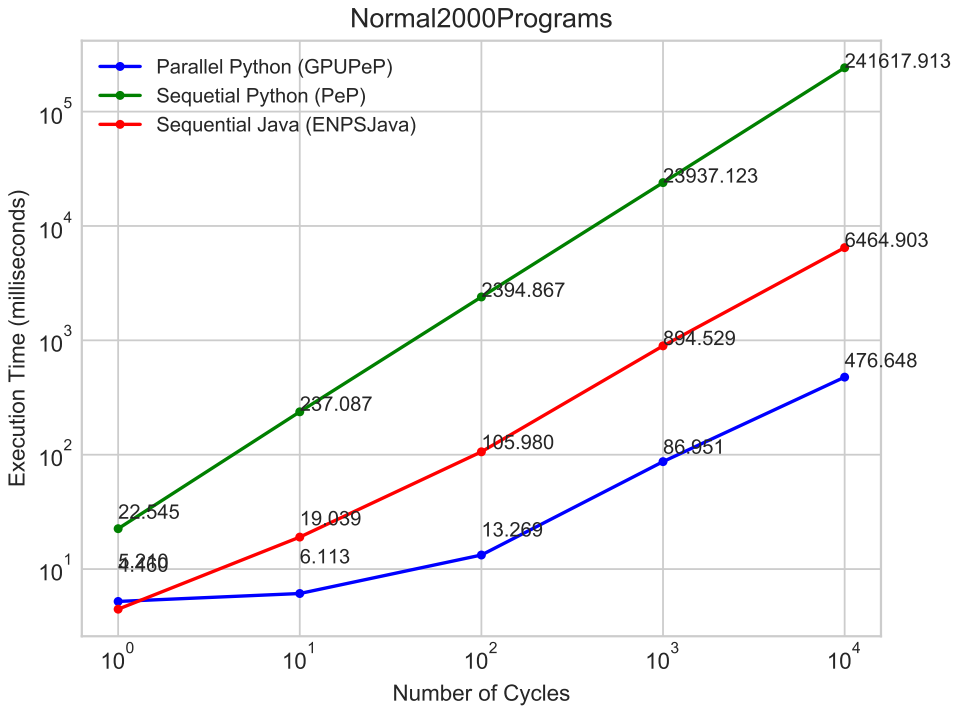

Figure 4.21 Normal 2000 Programs

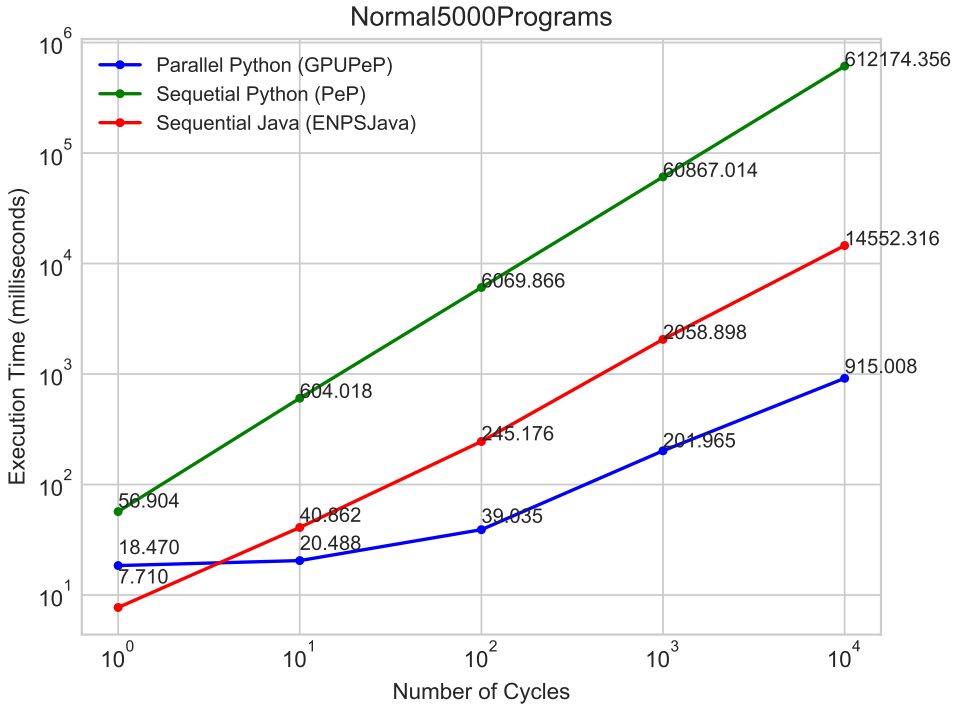

Figure 4.22 Normal 5000 Programs

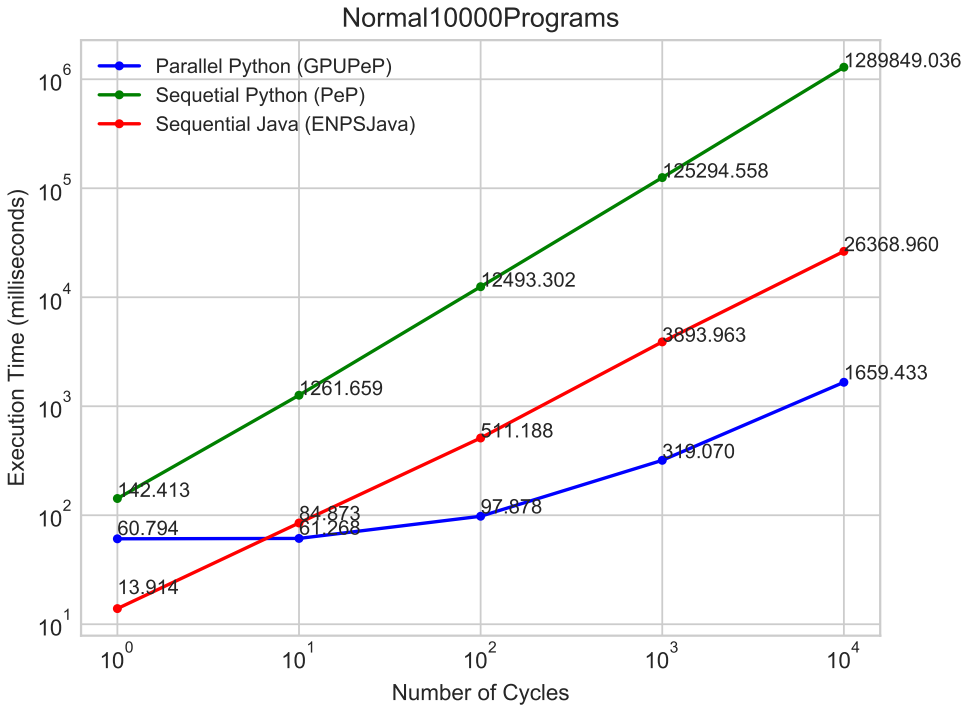

Figure 4.23 Normal 10000 Programs

precision. The float variant is faster than the variant double, with a trade-off loss in precision. Also, the double variant can only run on NVIDIA GPUs with compute capability $\geq$ 6.0 as *atomicAdd()*, a predefined function. For floating-point based double-precision numbers, it is not available on devices with compute capability lower than 6.0 (Nvidia, 2019). The results, are for float variants. Two variants are given to facilitate the user to choose, according to their needs.

Case study I

The first case study consists of a simple membrane which tests the ability of the system to handle large membranes of over 10000 programs. This also examines the efficiency of the tool with regard to parallelism using seed membranes (García Quismondo et al., 2012a). A seed membrane is a simple membrane that gives an abstract structure which can be transformed into a bigger membrane by replicating the operation (programs and membranes) performed by the seed membrane. The membrane, as shown in figure 5.7, is replicated and is accordingly analyzed further.

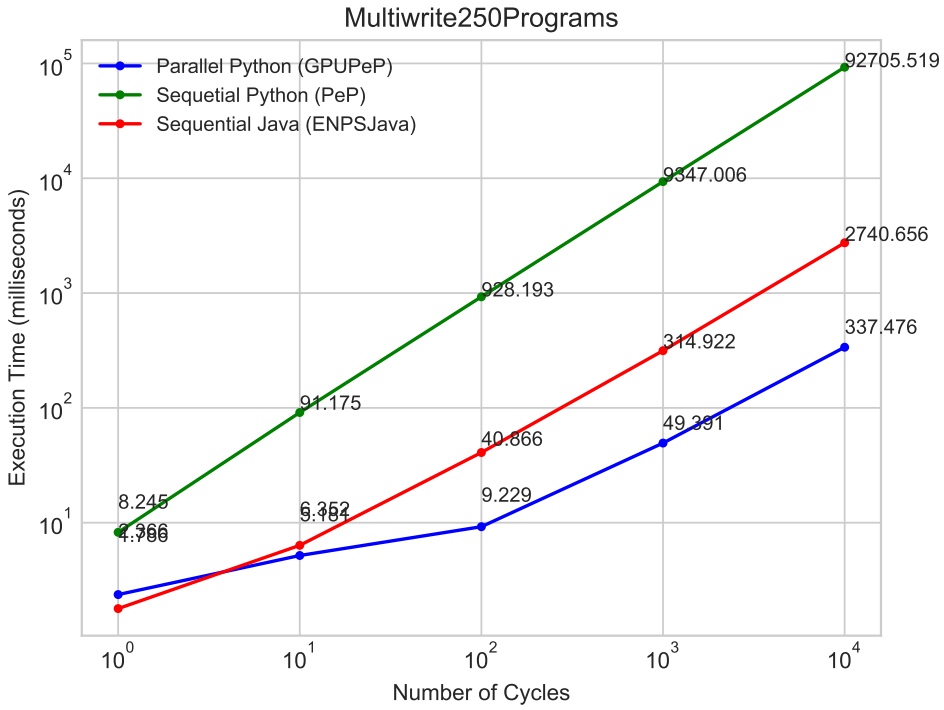

Figure 4.24 Multiwrite 250 Programs

There are four different sizes of programs considered (1000, 2000, 5000, 10000) to study the efficiency of a normal membrane system. These are executed in three different simulators (GPUPeP, PeP and ENPSJava). GPUPeP is executed on NVIDIA Tesla T4 GPU and ENPSJava is run on Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz. PeP has initially been executed on the Xeon CPU, but has obtained far better simulation times, running it on Intel(R) Core i7-4790 CPU @3.60 GHz, therefore, the results are better and comparable to the latter. All the three execution times are compared. To avoid environment related discrepancy, each membrane structure is executed 10 times and an average of all the executions on each simulator for each membrane structure, is considered.

From figures 4.20, 4.21, 4.22, 4.23, it is to be noted that GPUPeP is initially slower than both PeP and ENPSJava for 1 cycle in Normal 10000 Programs. Same is the case with other samples for 1000, 2000 and 5000 programs. However, as the number of cycles increases, GPUPeP catches up and has an average speedup of 770x and 15.8x, against PeP and ENPSJava respectively for 10000 cycles (for different cases).

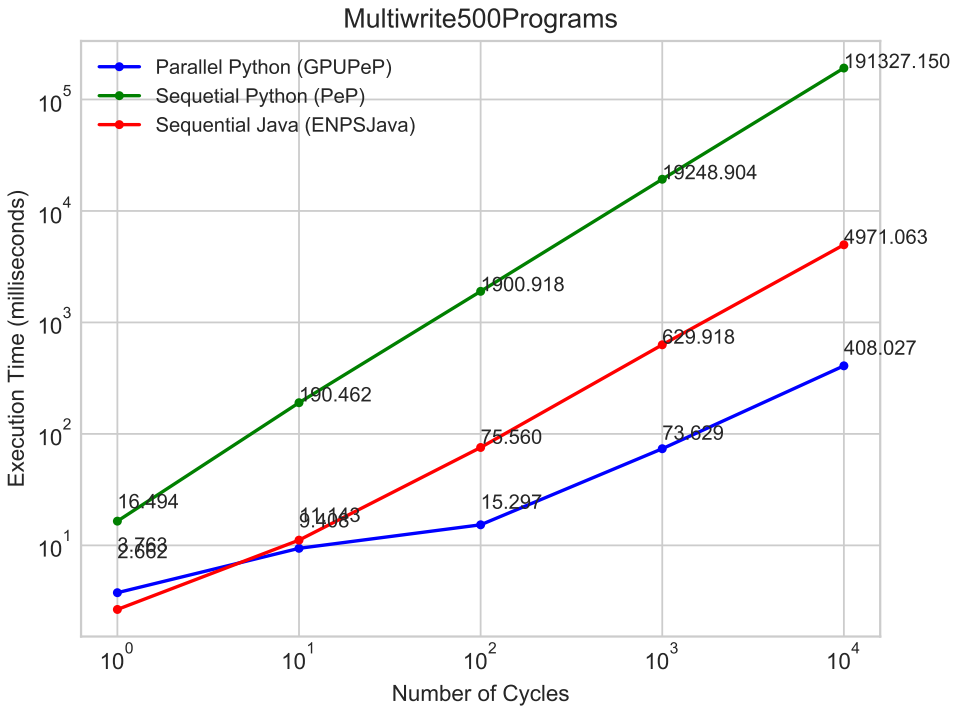

Figure 4.25 Multiwrite 500 Programs

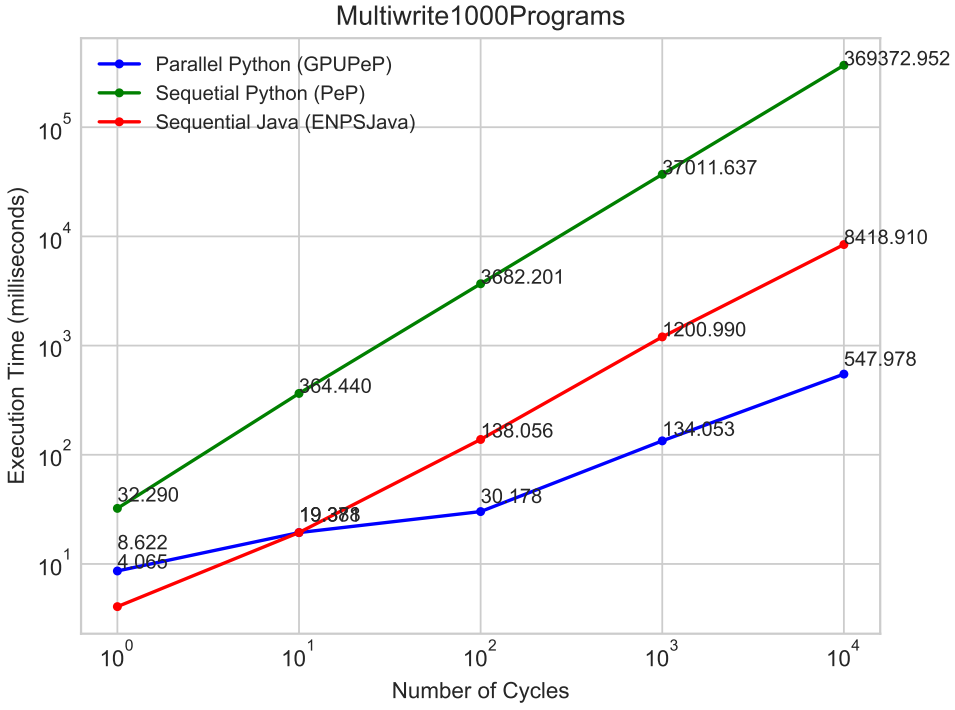

Figure 4.26 Multiwrite 1000 Programs

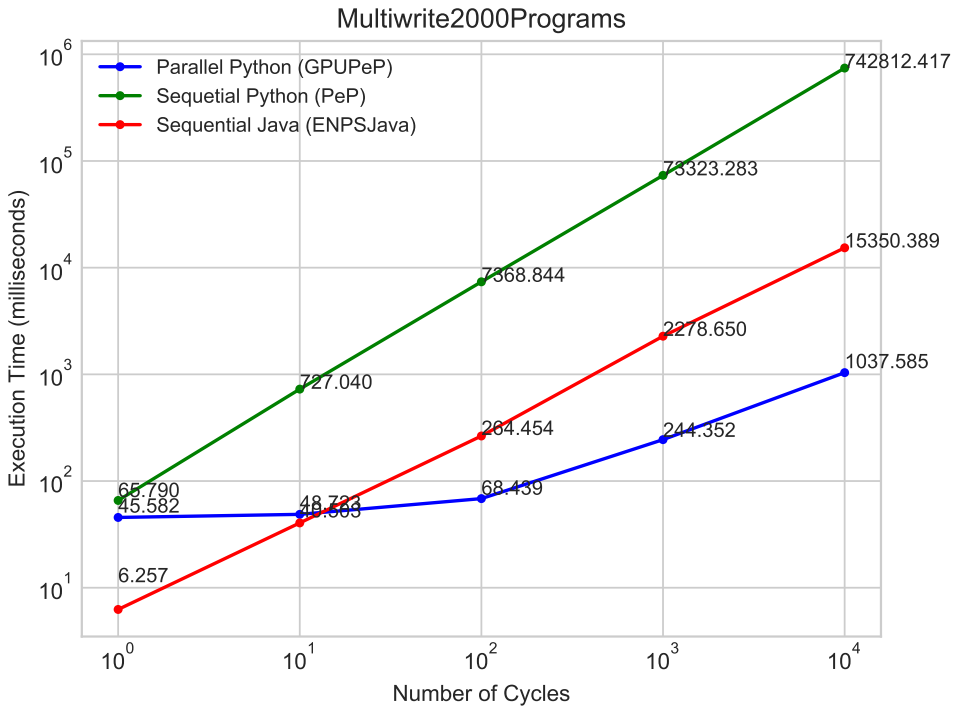

Figure 4.27 Multiwrite 2000 Programs

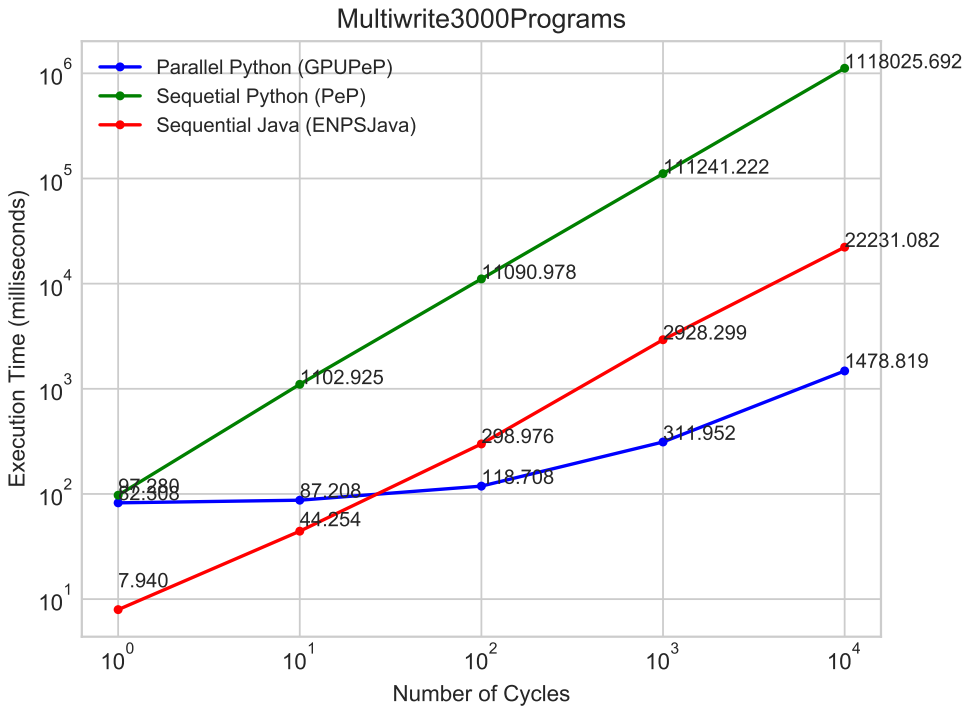

Figure 4.28 Multiwrite 3000 Programs

A CUDA kernel does the parallel simulation in GPUPeP. When the number of cycles to simulate is small (1 and 10), very less work is done inside the kernel. Thus, most of the time is spent on launching the CUDA kernel and, therefore, is relatively slow compared to PeP and ENPSJava. When the number of cycles to simulate is high (more than 10), sufficient time is spent in the CUDA kernel for parallel simulation. So, the computation time masks the initial kernel launch overhead. Therefore, GPUPeP has significant speed up against serial PeP and ENPSJava (CPU based). From the graphs, it can be observed that the membrane (Figure 4.22) with 5000 programs is the most efficient one, with an average execution time than the ENPSJava, by at least 15.9 times for 10000 cycles and 770x for PeP. The others are also at least 8 times better for 10000 cycles. Similarly, for 10000 programs, the GPUPeP is 770 times better than PeP for the best case.

The possible reason for this is the efficiency of the proposed architecture with the capabilities of the CUDA framework, to utilize the inherently powerful GPUs. Further, the reason for getting the speedup to nearly 770x, as compared to PeP, is primarily because of the design of PeP (Florea and Buiu, 2017, 2018). PeP, designed by Florea et al. (Florea and Buiu, 2017, 2018), is one of the first Python-based sequential simulator designed to give accurate results for any ENPS membrane input. Whereas, simulation efficiency doesn't seem to be a primary goal of PeP, as everything is executed sequentially in it. Thus, PeP gives accurate results, but takes considerable time when the input is in the range of several thousands of programs and cycles. Thus, GPUPeP gives a high computational speedup against PeP for these types of large executions.

Case study II

In this subsection, the system's performance is analyzed when a highly re-writable parallel membrane structure is given. Further, the capability of the system to run a large membrane structure is studied. For testing, a simple membrane seed structure is defined. The seed membrane structure consists of three membranes (with one shown as in Figure 4.16). The other two membranes have the same set of programs. The programs shown in the membranes are replicated multiple times to over 3000 programs (1000 for each membrane), to test the GPU's worst case effectiveness. To study the effect of a highly re-writable mem-

brane system, there are five different sizes of membranes considered, 250, 500, 1000, 2000, 3000 as in Figure 4.24, 4.25, 4.26, 4.27 and 4.28 respectively. These are executed in three different simulators (GPUPeP, PeP and ENPSJava). Here again, each membrane structure is executed 10 times and an average of all the executions on each simulator, for each membrane structure is considered. Initially, in multi-writable programs, GPUPeP is relatively slow due to kernel launch overhead. However, as the number of cycles increases, the GPUPeP has a better performance. Even in the worst case (Multi-write 3000 and Multi-write 1000), it has an average speedup of 750x and 15.3x against PeP and ENPSJava, respectively. Other cases are at least 8.1x better (250 programs provide very little scope for parallelism). Every other case is at least 11 times better.

4.3 Summary

Two Python-based tool for ENPS are developed for this study. The first tool is Multi-membrane system execution tool which supports two standard simulators (PeP and ENPSJava). The tool also supports file inter-conversion between two standard formats (.pep and .xml). This has been further tested and the results are analysed. Similarly, another tool named GPUPeP is developed, which is GPU based ENPS simulator. This simulator is found to be one of the best and ascertains that it is compared with two other standard simulators and the speedup results are obtained. The tools are used for specific application as per the requirements. The results show the tools to be performing perfectly according to the requirements, both forming an important asset.

Chapter 5

CLOUD SERVICE SELECTION USING P SYSTEM

Cloud service selection is a process of selecting the best service from a set of available cloud services. As the number of cloud services are increasing day by day there is a need for a service selection algorithm that can rank these services efficiently. This study proposes service selection mechanism based on a parallel computing paradigm variant ENPS coupled with a MCDM structure. There are two approaches that have been proposed here. As cloud services are involved our primary aim is to develop at a robust algorithm based.

5.1 Introduction

Service Selection in Cloud is one of the prime areas of research, within the ambit of cloud computing, that has gained quite a wide attention in the recent past. Service selection algorithm primarily involves selecting the best service from a set of available services based on Quality of Service (QoS) attributes. The QoS attributes are the parameters which allow the users to ascertain the actual quality of the service, usually quantitatively. Over the years there have been several methods designed for service selection in cloud, and primarily they are sequential in nature.

These QoS attributes would often test the favorability of the services to the user, based on the requirement and aim of the project. There are three components of a normal cloud

service selection model, the service consumers, the broker, and service providers (Sundareswaran et al., 2012; Liu et al., 2011) (as in figure 2.1). A service consumer is a user who leases services from the service providers. The broker acts as an interface between the service consumer and the service provider. Brokers allow consumers to access service providers. There are several service providers available; the broker handles communication with a set of service providers with several other value-added services (Liu et al., 2011; Equinix, 2019). An essential job of the broker is to facilitate service selection in cloud by allowing the user to select from the available set of services. It usually ranks services for selection. This is the most appropriate place for the service selection algorithm (ranking) to be present. As in the figure 2.1, the service selection algorithm is placed inside the broker. For example, Equinix (Equinix, 2019) a cloud service broker, is known to have more than 400 cloud service providers as its clients (Lin et al., 2016) and within these cloud service providers each might have more than atleast 4 services to offer. Thus a customer is allowed to select from approximately 1600 services (ranking 1600) which is quite a lot of services. Thus a effective approach which can run efficiently for several thousands of services as choices has to be designed.

Therefore an effective approach which can run efficiently for several thousands of services as choices has to be designed. In this scenario, there is always a possibility of small changes in the weights and values. Thus the proposed model should be able to bear the changes in the weight, i.e. the proposed model should be robust models which can bear slight changes in the weight and no effect in output are useful.

Considerable work has been done in this area, and each of the work has a specific advantage. A Literature Review has been done and it is found that almost all of the works, except few, consider a sequential model for designing the algorithm whose complexity increases non-linearly for a linear increase in the number of services available for selection. Thus, a robust parallel model for service selection in cloud is proposed. There are several methods to solve the problem of service selection in cloud, classified into several categories as Optimization-based approaches, Multi-criteria Decision Making (MCDM), Logic-based approach and other miscellaneous approaches (Sun et al., 2014). Out of these one of the most widely used is MCDM approaches. MCDM models are specifically designed for

ranking a set of items given the values of the attributes of the items and weights of the attributes.

Gheorghe Paun has introduced membrane Computing in his seminal paper in 2000 (Păun, 2000). The devices used to realize Membrane computing are called as P Systems. P System is inspired by nature, in particular, by a living cell. A membrane model has a hierarchical structure of membranes where the internal components are disjoint figure 2.2. The outer layer is called as skin membrane and all the other membranes are contained inside it. There can be any number of membranes present inside the skin and all these membranes can communicate with each other, i.e. they can pass information between them. This structure inherently supports parallelism, based on which the whole design works. Unlike other parallel models that have a limited scope or constrained scope of application, P System has a wider range of applications due to its being a computational paradigm. There are numerous variants of P Systems available (Paun et al., 2010). Each variant has a different structure and is designed for a different purpose. One such specific variant called Enzymatic Numerical P System (ENPS) is used for the chosen problem of Multi-criteria Decision Making (for cloud service selection). The primary contribution of the chapter as follows:

- A moderately sensitive, inherently parallel membrane-based MCDM approach for cloud service selection
- Another robust, inherently parallel membrane-based MCDM approach for cloud service selection
- An automated generator which allows an easy generation of these membrane modules

5.1.1 Service Selection in Cloud

There are several works in the area of cloud service selection. According to Sun et al. (Sun et al., 2014) there are primarily four classifications methods for cloud computing, MCDM-based approaches, Optimization-based approaches, Logic-based and other approaches. Of

the four approaches, the most popular one is Multi-Criteria based decision making. After 2014, there have been several works in the area of cloud service selection, specifically using MCDM methods. Some important and latest methods that have some relation to the proposed approach are discussed, primarily MCDM approaches. As parallelism and robustness, every method has some aim to be achieved through their proposed model and that has been discussed.

Many inherently parallel paradigms are available, like Molecular computing (DNA Computing, Membrane Computing, Peptide Computing), Amorphous Computing, Quantum Computing, to name a few (Kari and Rozenberg, 2008). Based on the structure of service selection problem (MCDM) and the mentioned models, we find membrane models to be suitable to solve this problem because of their properties like; simple inherent parallel structure, Turing Universality, multiple variants, variety of applications and its developing software support. In recent years there has been several direct application of membrane computing as membrane algorithms and there have been several variants of P system with Enzymatic Numeric P System being one.

5.1.2 Enzymatic Numerical P System (ENPS)

A primary structural variant of P System is a cell-like P system. The structure of a membrane system is as in figure 2.2. It consists of individual membranes placed in a hierarchical model (Păun, 2000) with the outermost membrane called a skin membrane. The skin membrane is a mandatory requirement for a membrane system. Inside the skin membrane, there can be nil to any number of child membranes. Every membrane has certain rules-based instructions called program (though name programs are specific to ENPS, we are generalizing it as it is analogous to multi-set rules defined for general P System). These programs are the basic building blocks that are responsible for realizing computing through membranes. All the child membranes have programs and the skin membrane may or may not have programs. The programs can pass information from one membrane to any other membrane directly.

An important property of P System is parallelism, which is exhibited in two ways;

Membrane level parallelism and Program level parallelism (Paun et al., 2010). Membrane level parallelism refers to the property where existing membranes execute in a parallel manner while its programs are sequential and Program level parallelism refers to the property where programs execute in parallel while the membranes execute sequentially. Here, both the modes work together and every program of every membrane executes parallelly. Based on the properties of the problem ENPS is used as solution. The details of ENPS structure are elaborated in section 2.4. Further, the exact solution of service selection (ranking) in cloud based on ENPS and MCDM is elaborated as follows.

5.2 Logical Operations behind ENPS-ITOPSIS

The proposed model (ENPS-ITOPSIS) involves certain logical operations that are made into membrane-based structures. The operations are discussed in sequence, so that, conceptually the service selection operation is understood more easily. Albeit, there is no step-wise mapping of the logic to the proposed approach, but all the operations are used in the method. These are based on the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) method (Hwang and Yoon, 1981; Rao, 2007), which has been developed by Hwang and Yoon (Hwang and Yoon, 1981) and further extended by Rao (Rao, 2007) as Improved TOPSIS and Deng et al. (Deng et al., 2000) as Modified TOPSIS.

The proposed ranking method is used, when a set of alternatives with attributes is given as input with weights. The cloud QoS attribute values are given for each alternative with the attributes corresponding weights given as input. The method is based on the concept that, the chosen alternative is nearest to the ideal solution and has to be the farthest to the negative ideal solution. The distance used is the Euclidean distance. An ideal solution is a hypothetical alternative, for which all the attributes are at their maximum i.e., the best value for each attribute. Similarly, a negative ideal solution is a value that corresponds to the alternative, which has the minimum values for the attributes i.e., the worst value for the attributes. There are primarily a set of operations executed in a sequence, that is elaborated and described as follows to get the final ranking:

The first step is to evaluate the given set of attributes and alternatives. These sets of

attributes and alternatives are given in the form of a matrix (Decision Table) where the columns represent the attributes, and the rows represent the alternatives. Each cell, in the decision table, is denoted as m_{ij}, where it represents the values of i^{th} alternative and j^{th} attribute. The values are real values ($\mathbb{R}$), which are in accordance to the type of attributes considered.

The given decision table consists of values according to the attribute ranges, as available in the raw form. This decision table has to be normalized so that it can be used for the next steps. The normalized decision is calculated using the formula given in equation 5.1

$$R_{ij} = m_{ij} \,/\, [\sum_{j=1}^{M} M_{ij}^2]^{0.5} \tag{5.1}$$

After the normalized values have been calculated, the weights of the attributes are required to proceed further. The weights are directly available, or according to Rao (Rao, 2007), IAHP can be considered for calculating the weights, provided the user gives a matrix with user preferences i.e., the order of preference for each attribute, against other attributes.

The weights are represented by w_i (for $i = 1, 2, \ldots, m$), such that, the sum of all the weights is 1 i.e. $\sum_{j=1}^{m} w_j = 1$. If the user directly gives the weight, the decision-maker can proceed with the next step, else the user can give the preference matrix. It consists of the relative importance of attributes and it is assigned, based on the user's experience and knowledge. The decision-maker, based on the values given by the user in the preference matrix, can use IAHP (Rao, 2007) for calculating the actual weight.

The weighted normalized matrix V_{ij} is calculated. This is done by multiplying each element of the matrix R_{ij} with their corresponding weights of the attributes w_j. The resultant matrix is v_{ij} which is expressed in equation 5.2:

$$V_{ij} = w_j \times R_{ij} \tag{5.2}$$

The ideal best solution and ideal worst solution are then calculated. The formula for this is given in equation 5.3, 5.4 (Rao, 2007) :

$$V^+ = (V_{ij}/j), (V_{ij}/j^{'}) \ \ for \ \ i = 1,2,\ldots,n. \tag{5.3}$$

$$V^- = (V_{ij}/j^{'}), (V_{ij}/j) \ \ for \ \ i = 1,2,\ldots,n. \tag{5.4}$$

Here there are two kinds of attributes, beneficial attributes, and non-beneficial attributes.

For the beneficial attributes, the higher the attribute value, the better the impact and conversely, for the non-beneficial attribute, the lower the value, the better the impact. (The maximum and minimum values are calculated accordingly). The beneficial attribute has the numerically highest value and that is considered for V^+, similarly, numerically lowest value is considered for V^-. This is reversed in the case of the non-beneficial attribute, as the numerically lowest value is considered for V^+ and a numerically higher value is considered for V^-.

The next step is to obtain separation measures. The positive ideal separation measure is the Euclidean distance between the ideal solution (equation 5.6) and an alternative. The negative separation measure is the euclidean distance between the negative ideal solution and alternatives (equation 5.5). To get the rank, the value of the final separation measure, for each alternative, is sorted in descending order (in equation 5.7) (Rao, 2007).

$$S_i^+ = [\sum_{j=1}^{M} (V_{ij} - V_j^+)^2]^{0.5} \tag{5.5}$$

$$S_i^- = [\sum_{j=1}^{M} (V_{ij} - V_j^-)^2]^{0.5} \tag{5.6}$$

$$P_i = S_i^- \div (S_i^- + S_i^+) \tag{5.7}$$

5.3 Membrane Based Improved Technique for Order of Preference by Similarity to Ideal Solution

As mentioned in figure 2.1, the membrane-based approach is placed under cloud service selection module. Here, cloud service selection refers to the method where, given the details about services, a ranking of services is generated as output and the best one is selected by the user. This even applies for the proposed membrane-based technique, this applies. The details related to the services, like QoS Parameters and their corresponding weights (either calculated or already available), are considered as input. The proposed method takes this input and gives the ranks as output, as shown in figure 5.2.

Membrane-based algorithms proposed here, strictly follow the membrane structure and the rules associated with the model. The whole process of ranking involves the collection of four membrane systems (membranes), that execute as per the membrane computing protocol in a certain sequence (as given in figure 5.1). This entire structure, collectively as a single entity is the proposed algorithm; the input to this membrane module (algorithm) is the services detail (QoS Parameters and their weights) and the output is the rank of the services. Details about each of the membrane systems, mentioned in figure 5.1, are given with subsequent discussions, later in this section. These membrane systems refer to a single membrane or collection of membranes which execute and generate output, based on

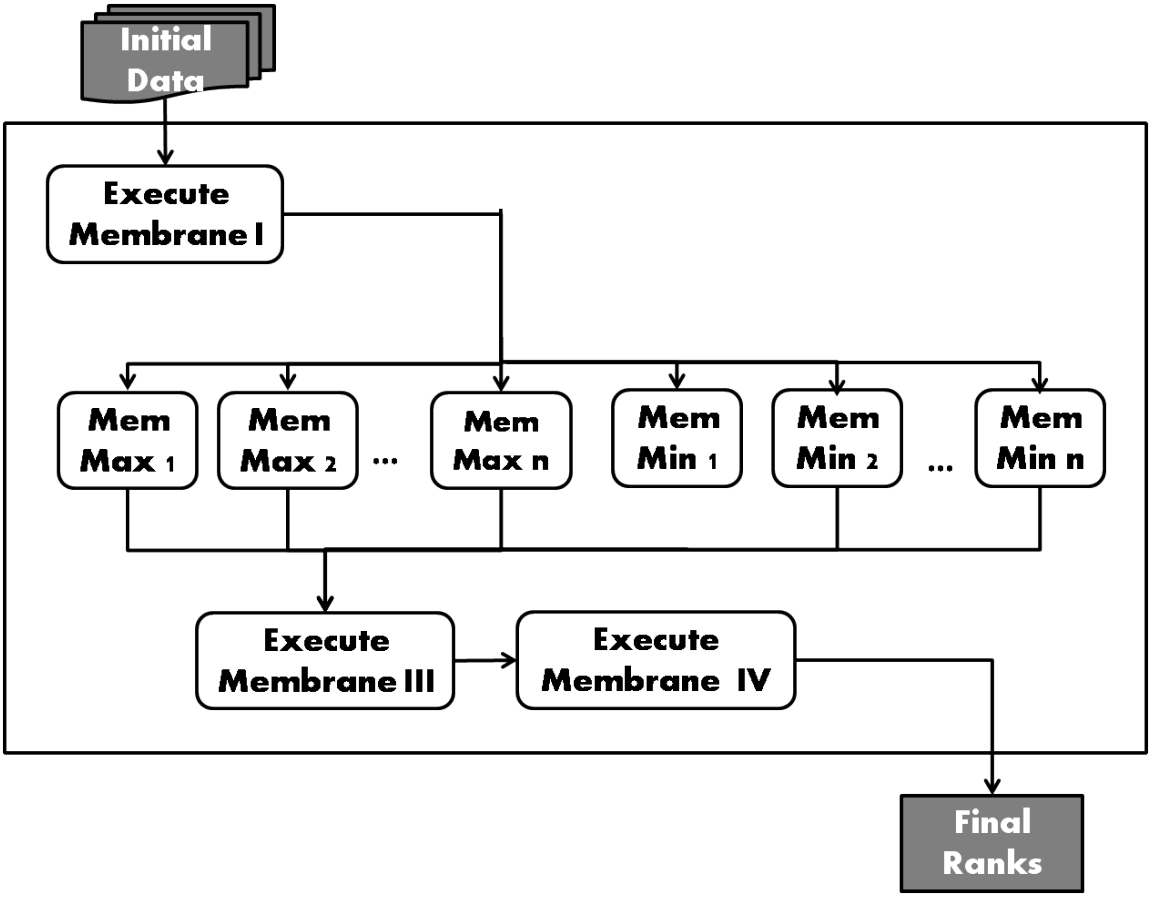

Figure 5.1 ENPS-ITOPSIS Structure

the input passed to it, after executing for a number of cycles that have been specified. Thus, the whole process of ranking involves four membrane systems, each performing different tasks. Subsequently these have been elaborately discussed.

There are two variants of the models that have been proposed in the work. Both the variants consist of four membrane systems, and have the same logical flow between the membranes, as given in figure 5.1. The only difference between these models is the rules in membrane 1 and membrane 4, which are elaborated later in this section. The first model ENPS-ITOPSIS is discussed in detail, later pointing out the small changes that are made, to get the other variant called ENPS-MTOPSIS. Both the variants work in a broker environment, specifically in the service selection module, where details related to the services, like QoS values for attributes and weights are given as inputs to the Membrane module (algorithm).

The first and primary membrane-based algorithm proposed in this study is Enzymatic Numerical P System - Improved Technique for Order of Preference by Similarity to Ideal Solution (ENPS-ITOPSIS). It consists of four steps, as described in figure 5.2. The execution model is given in figure 5.1 The operations performed in ENPS-ITOPSIS are similar as in ITOPSIS (Rao, 2007), but the base and methodology are totally different. These are performed to suit the membrane structure, which is inherently parallel and has a membrane computing structure. The first step of the method is to normalize the given data into the proper format.

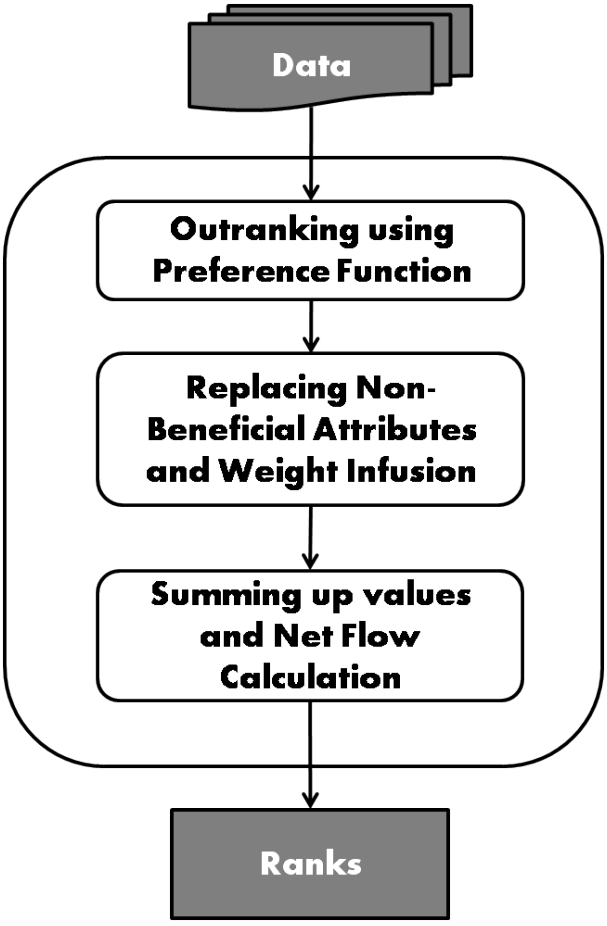

Figure 5.2 ENPS-ITOPSIS execution flow

Step 1- Membrane System 1- Normalization:

The first step (figure 5.1 denotes Membrane1 and its logical operation denotes the first step in figure 5.2) involves normalization of the values, as shown in membrane 1 (figure 5.3). Here the normalization process using ENPS model, as in figure 5.3, is discussed in

detail.

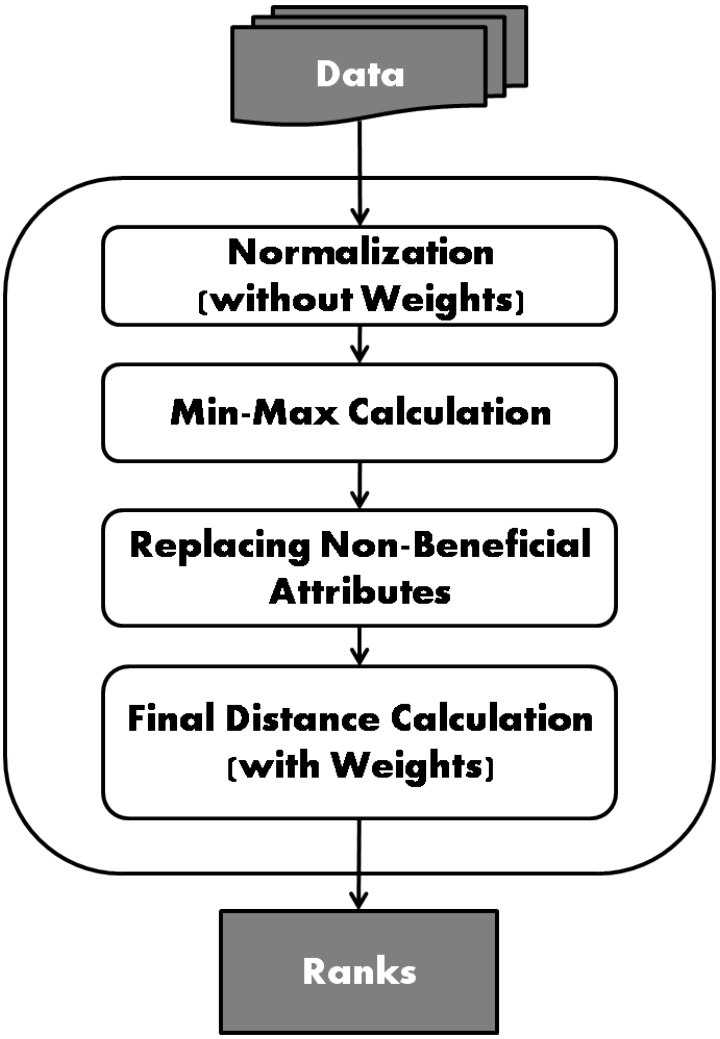

Figure 5.3 ENPS-MTOPSIS execution flow

As discussed in the definition of ENPS, each membrane has programs (understood as statements) with the enzymes (understood as controllers) which execute parallelly, for a certain number of cycles, to give the result. The same rules of ENPS are followed (as in section 3) and the normalization process is again broken into several steps that have been elaborated as follows:

The first step of normalization involves taking the square root of the sum of squares of each alternative. This is performed by the whole set of programs from $Pr_{1,1}$ to $Pr_{n,1}$, for n alternatives, and this is controlled by a single enzyme. All the programs are active in the current cycle and execute parallelly.

The next set of programs are from Pr_{n+1} to $Pr_{nm+n,1}$ which are used to retain the values of the alternatives (for m attributes) for the next cycle. These values are to be retained, so as to be used in the next cycle, for the final step of normalization. The retained values are divided by the values calculated in the previous cycle ($k_{1,1}$ to $k_{n,1}$). In the current cycle, the weights are multiplied and the weighted normalized values are obtained. These weighted normalized values obtained are passed on to the next membrane system.

$\underline{Membrane - System\,1\; -\; M_1}$

$$x_{1,1}[value], \ldots, x_{n,1}[value] \ldots\ldots z_{1,1}[value], \ldots, z_{n,1}[value]$$
$$k_{1,1}[value], \ldots, k_{n,1}[value]$$
$$m_{1,1}[value], \ldots\ldots, m_{n,1}[value] \cdots p_{1,1}[value], \ldots, p_{n,1}[value]$$
$$w_{1,1}[value], \ldots, w_{n,1}[value], e_1[l], e_2[0]$$

$$Pr_{1,1} : [x_{1,1}^2 + \ldots + z_{1,1}^2]^{0.5} \;\langle e_1 \rightarrow\rangle \;1 \mid k_{1,1}$$
$$\cdots$$
$$Pr_{n,1} : [x_{n,1}^2 + \ldots + z_{n,1}^2]^{0.5} \;\langle e_1 \rightarrow\rangle \;1 \mid k_{n,1}$$

$$Pr_{n+1,1} : x_{1,1} \;\langle e_1 \rightarrow\rangle \;1 \mid x_{1,1}$$
$$\cdots$$
$$Pr_{n+m,1} : z_{1,1} \;\langle e_1 \rightarrow\rangle \;1 \mid z_{1,1}$$
$$\cdots$$
$$Pr_{nm-m+n,1} : x_{n,1} \;\langle e_1 \rightarrow\rangle \;1 \mid x_{n,1}$$
$$\cdots$$
$$Pr_{nm+n,1} : z_{n,1} \;\langle e_1 \rightarrow\rangle \;1 \mid z_{n,1}$$
$$Pr_{nm+n+1,1} : x_{1,1} \div k_{1,1} \times w_{1,1} \;\langle e_2 \rightarrow\rangle \;1 \mid m_{1,1}$$
$$\cdots$$
$$Pr_{nm+n+m,1} : z_{1,1} \div k_{1,1} \times w_{1,1} \;\langle e_2 \rightarrow\rangle \;1 \mid m_{m,1}$$
$$\cdots$$
$$\cdots$$
$$Pr_{2nm+n-m,1} : x_{n,1} \div k_{n,1} \times w_{n,1} \;\langle e_2 \rightarrow\rangle \;1 \mid o_{1,1}$$
$$\cdots$$
$$Pr_{2nm+n,1} : z_{n,1} \div k_{n,1} \times w_{n,1} \;\langle e_2 \rightarrow\rangle \;1 \mid o_{m,1}$$
$$Pr_{2nm+n+1,1} : l \;\rightarrow\; 1 \mid e_2$$

Figure 5.4 Membrane System for Step 1

Step 2-Membrane System 2- Maximum and Minimum Value Calculation:

This is the second logical step, as in figure 5.2). This step involves the calculation of minimum and maximum values for each set of attributes (for all alternatives). Each set of attributes is divided and sent to the membrane; as mentioned in figure 5.1. This membrane calculates the minimum values. For any number of alternatives, the membrane executes only for three cycles. The membrane structure involves several enzymes for controlling the programs. The first cycle involves assigning values to the enzymes, which eventually control programs in the second cycle. All of these enzymes are selectively active, based on the values of the variables in the program. In cycle one, the set of enzyme control two programs each. The first one is for passing the minimum value and the other is for keeping track of the occurrence of each value. A track is kept of the occurrence of each value, in order to avoid error of duplicate values being present. The third enzyme is used to get the final minimum value after dividing the cumulative value obtained, with the number of occurrences.

The calculation of maximum value is similar to the calculation of minimum values, with a few differences, as in figure 5.5. The first set for cycle one involves the distribution of values to the enzyme, after subtracting the current values from a sufficiently high number. The next step involves all the programs, which are activated and the corresponding program calculates the needed cumulative value, while keeping track of the number of occurrences of each value. After the current cycle, these values are again subtracted from the larger number and then divided by the number of occurrences. This results in giving the maximum value in the given list.

$$\underline{Membrane - System 2 - M_1(Minimum)}$$

$$k_{1,1}[value], \ldots k_{m,1}[value],\ k_{m+1}[0]$$
$$e_1[0], \ldots,\ e_m[0],\ e_{m+1,1}[value], e_{m+2,1}[-l]$$

$$Pr_{1,1}:\ 2 \times k_{1,1}\ \ \langle e_{m+1,1} \rightarrow \rangle\ \ 1 \mid e_{1,1} + 1 \mid k_{m+1,1}$$
$$\cdots$$
$$\cdots$$
$$Pr_{m,1}:\ 2 \times k_{m,1}\ \ \langle e_{m+1} \rightarrow \rangle\ \ 1 \mid e_{m,1} + 1 \mid k_{m+1,1}$$

$$Pr_{m+1,1} : -1 \times k_{1,1} + k_{1,2} \times 0 + \cdots + k_{m,1} \times 0 \langle e_{1,1} \rightarrow \rangle 1 \mid k_{m+1,1}$$
$$Pr_{m+2,1} : k_{1,1} \times 0 + k_{1,2} \times -1 + \cdots + k_{m,1} \times 0 \langle e_{2,1} \rightarrow \rangle 1 \mid k_{m+1,1}$$
$$\cdots$$
$$Pr_{2m,1} : k_{1,1} \times 0 + \cdots + k_{m-1,1} \times 0 + k_{m,1} \times -1$$

$$\langle e_{m,1} \rightarrow \rangle 1 \mid k_{m+1,1}$$
$$Pr_{2m+1,1} : k_{1,1} \times 0 + \cdots + k_{n-1,1} \times 0 + k_{m,1} \times 0 + k_{m+2,1} \div k_{m+2,1} \times -1$$

$$\langle e_{1,1} \rightarrow \rangle 1 \mid k_{m+3,1}$$
$$\cdots$$
$$Pr_{3m,1} : k_{1,1} \times 0 + \cdots + k_{n-1,1} \times 0 + k_{m,1} \times 0 + k_{m+2,1} \div k_{m+2,1} \times -1$$

$$\langle e_{m,1} \rightarrow \rangle 1 \mid k_{m+3,1}$$
$$Pr_{3m+1,1} : l \langle e_{m+1,1} \rightarrow \rangle 1 \mid e_{m+2,1}$$
$$Pr_{3m+2,1} : k_{m+1,1} \div k_{m+3,1} \langle e_{m+2} \rightarrow \rangle 1 \mid k_{m+1,1}$$
$$Pr_{3m+3,1} : k_{m+2,1} \langle e_{m+1} \rightarrow \rangle 1 \mid k_{m+2,1}$$

Figure 5.5 Membrane System for Step 2 - Finding Minimum

Step 3 - Membrane System 3 - Replacing Non-Beneficial Attributes:

This the third logical step as given in figure 5.2. Once the maximum and minimum values for each attribute (for all alternatives) are obtained, the values are interchanged according to their category. There are two primary categories of attributes; beneficial attributes and non-beneficial attributes. For beneficial attributes, the higher the value, the better is the solution. For non-beneficial, the lower the value, the better is the solution. This is an important portion of the approach, where the proper distinction between these two kinds of attributes is made, based on which the method is able to assign the correct ranking. The current membrane does this job of interchanging the values parallelly. There are n enzymes to control $2n$ programs, where n is the number of attributes. The enzymes are assigned binary values, either 0 or 1. The value 0 indicates the corresponding attribute to be non-beneficial and attribute 1 indicates the values to be beneficial. The enzyme value as-

sures that only the non-beneficial values interchange. Thus, accordingly, the minimum and maximum values are interchanged and the final list obtained, consists of an ideal solution and negative ideal solution.

$$\underline{Membrane - System2 - M_1(Maximum)}$$

$$k_{1,1}[value], \ldots, k_{m,1}[value],\ k_{m+1,1}[0],\ k_{m+2,1}[0], k_{m+3}[0]$$

$$e_{1,1}[0],\ \ldots,\ e_{m,1}[0],\ e_{m+1,1}[0],\ e_{m+2,1}[-l],\ e_{m+3,1}[2l-1],\ n1[value]$$

$$Pr_{1,1}:\ 2\times[n1-k_{1,1}]\ \ \langle e_{m+1}\rightarrow\rangle\ \ 1 \mid e_1+1 \mid k_{m+1,1}$$

$$\cdots$$

$$\cdots$$

$$Pr_{m,1}:\ 2\times[n1-k_{m,1}]\ \ \langle e_{m+1,1}\rightarrow\rangle\ \ 1 \mid e_{m,1}+1 \mid k_{m+1,1}$$

$$Pr_{m+1,1}: e_{1,1}\times -1+e_{1,2}\times 0+\cdots+e_{m,1}\times 0\,\langle e_{1,1}\rightarrow\rangle 1 \mid k_{m+1,1}$$

$$Pr_{m+2,1}: e_{1,1}\times 0+e_{1,2}\times -1+\cdots+e_{m,1}\times 0\,\langle e_{2,1}\rightarrow\rangle 1 \mid k_{m+1,1}$$

$$\cdots$$

$$\cdots$$

$$Pr_{2m,1}: e_{1,1}\times 0+\cdots+e_{m-1,1}\times 0+e_{m,1}\times -1\,\langle e_{m,1}\rightarrow\rangle 1 \mid k_{m+1,1}$$

$$Pr_{2m+1,1}: e_{1,1}\times 0+e_{1,2}\times 0+\cdots+e_{m,1}\times 0+k_{m+2,1}\div k_{m+2,1}\times -1\langle e_{1,1}\rightarrow\rangle 1 \mid k_{m+3,1}$$

$$\cdots$$

$$\cdots$$

$$Pr_{3m,1}: e_{1,1}\times 0+\cdots+e_{m-1,1}\times 0+e_{m,1}\times 0+$$

$$k_{m+2,1}\div k_{m+2}\times -1\langle e_{m,1}\rightarrow\rangle 1 \mid k_{m+3,1}$$

$$Pr_{3m+1,1}: e_{m+1,1}-e_{m+1,1}\,\langle e_{m+1,1}\rightarrow\rangle 1 \mid e_{m+1,1}$$

$$Pr_{3m+2,1}: ((k_{m+3,1}*k_{m+2,1})-k_{m+1,1})/k_{m+3,1}\,\langle e_{m+2,1}\rightarrow\rangle 1 \mid k_{m+1,1}$$

$$Pr_{3m+3,1}: l\,\langle e_{m+3,1}\rightarrow\rangle 1 \mid e_{m+2,1}$$

$$Pr_{3m+4,1}: k_{m+2,1}\,\langle e_{m+3,1}\rightarrow\rangle 1 \mid k_{m+2,1}$$

Figure 5.6 Membrane System for Step 2 - Finding Maximum

Step 4 - Membrane System 4 - Final Distance Calculation:

The final logical portion is executed. With the interchanged value, the control proceeds towards the next membrane, the final membrane, where the distance between each attribute value of an alternative and the ideal solution is obtained. The designed membrane are as

mentioned in the figure 5.7. Enzyme e_1 controls the first set of programs, which is used to find the Euclidean distance between each of the value and its corresponding maximum and minimum value. The corresponding programs range from $Pr_{1,1}$ to $Pr_{m,1}$, for calculating the distance between max value and actual value. From $Pr_{m+1,1}$ to $Pr_{2m,1}$ the distance between the actual value and the minimum value is calculated. The rest of the programs are used for final distance calculation, except for $Pr_{3m+1,1}$, which is for enzyme control.

$$\underline{Membrane - System 3 - Membrane 1}$$

$$min_{1,1}[value], min_{2,1}[value], \cdots, min_{n,1}[1],$$
$$max_{1,1}[value], max_{2,1}[value], \cdots, max_{n,1}[value],$$
$$e_{1,1}[0 \mid 1], e_{2,1}[0 \mid 1] \cdots, e_{n,1}[0 \mid 1]$$

$$Pr_{1,1} : min_{1,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid max_{1,1}$$
$$Pr_{2,1} : min_{2,1} \ \langle e_{2,1} \rightarrow \rangle \ 1 \mid max_{2,1}$$
$$\cdots$$
$$Pr_{n,1} : min_{n,1} + 1 \ \langle e_{n,1} \rightarrow \rangle \ 1 \mid max_{n,1}$$
$$Pr_{n+1,1} : max_{1,1} \ \langle e_{1,1} \rightarrow \rangle \ 1 \mid min_{1,1}$$
$$Pr_{n+2,1} : max_{2,1} \ \langle e_{2,1} \rightarrow \rangle \ 1 \mid min_{2,1}$$
$$\cdots$$
$$Pr_{2n,1} : max_{n,1} + 1 \ \langle e_{n,1} \rightarrow \rangle \ 1 \mid min_{n,1}$$

Figure 5.7 Membrane for Step 3

The next enzyme controls the next cycle of execution, where the difference between the negative ideal solution and the ideal solution is calculated. These sets of programs are completely controlled by the enzyme e_2. Finally, the calculated distance is stored in a sequence of values from $d_{1,1} \cdots d_{n,1}$, where the values $1 \cdots n$ are corresponding to the values of the alternatives. Finally, these distance values are sorted to obtain the ranks, and the best among them is selected.

$\underline{Membrane - System 4 - M_1}$

$$x_{1,1}[value], \ldots, x_{n,1}[value] \ldots\ldots z_{1,1}[value], \ldots, z_{n,1}[value]$$
$$min_{1,1}[value], \ldots, min_{n,1}[value],\ max_{1,1}[value], \ldots, max_{n,1}[value]$$
$$l_{1,1}[value], \ldots, l_{n,1}[value],\ h_{1,1}[value], \ldots, h_{n,1}[value]$$
$$d_{1,1}[value], \ldots, d_{n,1}[value], e_1[l], e_2[0]$$

$$Pr_{1,1} :\ [[x_{1,1} - max_{1,1}]^2 + \ldots + [x_{n,1} - max_{n,1}]^2]^{0.5}\ \ \langle e_1 \rightarrow \rangle\ \ 1 \mid l_{1,1}$$
$$\ldots$$
$$\ldots$$
$$Pr_{m,1} :\ [[z_{1,1} - max_{1,1}]^2 + \ldots + [z_{n,1} - max_{n,1}]^2]^{0.5}\ \ \langle e_1 \rightarrow \rangle\ \ 1 \mid l_{m,1}$$

$$Pr_{m+1,1} :\ [[x_{1,1} - min_{1,1}]^2 + \ldots + [x_{n,1} - min_{n,1}]^2]^{0.5}\ \ \langle e_1 \rightarrow \rangle\ \ 1 \mid h_{1,1}$$
$$\ldots$$
$$\ldots$$
$$Pr_{2m,1} :\ [[z_{1,1} - min_{1,1}]^2 + \ldots + [z_{n,1} - min_{n,1}]^2]^{0.5}\ \ \langle e_1 \rightarrow \rangle\ \ 1 \mid h_{m,1}$$
$$Pr_{2m+1,1} :\ h_{1,1} - l_{1,1}\ \ \langle e_2 \rightarrow \rangle\ \ 1 \mid d_{1,1}$$
$$\ldots$$
$$\ldots$$
$$Pr_{3m,1} :\ h_{m,1} - l_{m,1}\ \ \langle e_2 \rightarrow \rangle\ \ 1 \mid d_{n,1}$$
$$Pr_{3m+1,1} :\ l\ \ \rightarrow\ \ 1 \mid e_2$$

Figure 5.8 Membrane System for Step 4

ENPS-MTOPSIS:

Though a simple change, there is a considerable effect in the results obtained for same input. Adding weight just before the distance (Step 4), tends to be more practical in few cases, thereby giving more better results related to robustness.

$\underline{Sub-problem1\ -\ M_1(Modified)}$

$$x_{1,1}[value],\ldots,x_{n,1}[value].....z_{1,1}[value],\ldots,z_{n,1}[value]$$
$$k_{1,1}[value],\ldots,k_{n,1}[value],e_1[l],e_2[0]$$
$$m_{1,1}[value],.....,m_{n,1}[value]\cdots p_{1,1}[value],\ldots,p_{n,1}[value]$$

$$Pr_{1,1}:\ [x_{1,1}^2+\ldots+z_{1,1}^2]^{0.5}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid k_{1,1}$$
$$\cdots$$
$$Pr_{n,1}:\ [x_{n,1}^2+\ldots+z_{n,1}^2]^{0.5}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid k_{n,1}$$
$$Pr_{n+1,1}:\ x_{1,1}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid x_{1,1}$$
$$\cdots$$
$$Pr_{n+m,1}:\ z_{1,1}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid z_{1,1}$$
$$\cdots$$
$$Pr_{nm-m+n,1}:\ x_{n,1}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid x_{n,1}$$
$$\cdots$$
$$Pr_{nm+n,1}:\ z_{n,1}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid z_{n,1}$$
$$Pr_{nm+n+1,1}:\ x_{1,1}\div k_{1,1}\ \ \langle e_2\rightarrow\rangle\ \ 1\mid m_{1,1}$$
$$\cdots$$
$$Pr_{nm+n+m,1}:\ z_{1,1}\div k_{1,1}\ \ \langle e_2\rightarrow\rangle\ \ 1\mid m_{m,1}$$
$$\cdots$$
$$\cdots$$
$$Pr_{2nm+n-m,1}:\ x_{n,1}\div k_{n,1}\ \ \langle e_2\rightarrow\rangle\ \ 1\mid o_{1,1}$$
$$\cdots$$
$$Pr_{2nm+n,1}:\ z_{n,1}\div k_{n,1}\times w_{n,1}\ \ \langle e_2\rightarrow\rangle\ \ 1\mid o_{m,1}$$
$$Pr_{2nm+n+1,1}:\ l\ \ \rightarrow\ \ 1\mid e_2$$

Figure 5.9 Membrane System for Step 1 (Modified) - ENPS-MTOPSIS

Conceptually, each membrane system, in all the four steps (for both ENPS-ITOPSIS and ENPS-MTOPSIS), execute a maximum for 3 cycles for any input. As the model is maximally parallel, each cycle has a complexity of O(1) and as the number of cycles doesn't increase beyond a constant (3), the complexity of each membrane execution is c O(1). Four steps are sequential because of the problem in hand, this also is constant and hence, the complexity is still c O(1). This is far better than any other sequential approach, which has a complexity of O(n^2) for these problems.

$\underline{Sub-problem4-M_1(Modified)}$

$$x_{1,1}[value],\ldots,x_{n,1}[value].....z_{1,1}[value],\ldots,z_{n,1}[value]$$
$$min_{1,1}[value],\ldots,min_{n,1}[value],\ max_{1,1}[value],\ldots,max_{n,1}[value]$$
$$l_{1,1}[value],\ldots,l_{n,1}[value],\ h_{1,1}[value],\ldots,h_{n,1}[value]$$
$$d_{1,1}[value],\ldots,d_{n,1}[value],w_{1,1}[value],\ldots,w_{n,1}[value],e_1[l],e_2[0]$$
$$Pr_{1,1}:\ [[x_{1,1}-max_{1,1}]^2\times w_{1,1}+\ldots+[x_{n,1}-max_{n,1}]^2\times w_{n,1}]^{0.5}$$
$$\langle e_1\rightarrow\rangle.\ 1\mid l_{1,1}$$
$$\ldots$$
$$Pr_{m,1}:\ [[z_{1,1}-max_{1,1}]^2\times w_{1,1}+\ldots+[z_{n,1}-max_{n,1}]^2\times w_{n,1}]^{0.5}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid l_{m,1}$$
$$Pr_{m+1,1}:\ [[x_{1,1}-min_{1,1}]^2\times w_{1,1}+\ldots+[x_{n,1}-min_{n,1}]^2 w_{n,1}]^{0.5}$$
$$\langle e_1\rightarrow\rangle\ \ 1\mid h_{1,1}$$
$$\ldots$$
$$\ldots$$
$$Pr_{2m,1}:\ [[z_{1,1}-min_{1,1}]^2\times w_{1,1}+\ldots+[z_{n,1}-min_{n,1}]^2\times w_{n,1}]^{0.5}\ \ \langle e_1\rightarrow\rangle\ \ 1\mid h_{m,1}$$
$$Pr_{2m+1,1}:\ h_{1,1}-l_{1,1}\ \ \langle e_2\rightarrow\rangle\ \ 1\mid d_{1,1}$$
$$\ldots$$
$$\ldots$$
$$Pr_{3m,1}:\ h_{m,1}-l_{m,1}\ \ \langle e_2\rightarrow\rangle\ \ 1\mid d_{n,1}$$
$$Pr_{3m+1,1}:\ l\ \ \rightarrow\ \ 1\mid e_2$$

Figure 5.10 Membrane System for Step 4 (Modified) -ENPS-MTOPSIS

5.4 Implementation

The proposed solutions are membrane-based MCDM models, for cloud service selection; there are two important points to be considered: implementing a strict ENPS model and applying it to a cloud service model. To ensure it uses strict ENPS as its base structure, the designed ENPS model is implemented using a python based simulator, PeP 3.0, developed by Florea et al. (Florea and Buiu, 2018). This simulator, simulates a whole ENPS membrane structure with proper output, with a membrane structure given as input. As only membrane models can be given as input, the researched model requires an automatic file generator, working over the PeP simulator, which can run the ENPS-MTOPSIS model, (directly over the simulator) to obtain the results. Hence, a generator named TP-generator is developed, using python. It is tailored with the PeP simulator to give the final ranks of the items, when given the weights and values of the attributes. The complete implementation

consists of four components; all of them designed to exactly map into the logical model being developed, as in figure 5.1. This generator generates all the membranes, given the alternatives and attribute data with the weights. Further, a GPU based simulator called GPUPeP (similar to PeP) is also used to take advantage of parallelism. As this approach is for cloud service selection, a real-time dataset collected by Sun et al. (Sun et al., 2019) is considered. It consists of attribute related details about the cloud services and is passed to the proposed membrane systems (ENPS-ITOPSIS and ENPS-MTOPSIS). The exact details about the dataset are as follows.

Dataset:

The details of the attributes are given in table 5.1. The dataset is taken from the work of Sun et al. (Sun et al., 2019) who have collected the datasets by continuous observation of several cloud services. In this work 15 services ($s1 - s15$) are considered with eight attributes (as given in table 5.1). The service names and values have not been shown to maintain privacy in this case. However, a sample of how each service data looks like is given in table 5.2. There are two required values for calculating the ranks of a given data (for service selection); the weights of the values, and the actual values of the attributes corresponding to each alternative (made public by (Sun et al., 2019)). This work does not take weights from the dataset (as different weights can be assigned) but are calculated based on the knowledge about the attributes and their properties.

The weight calculation part is a pre-processing step and is not part of the proposed model. If weights are proper as per the user, the selector may directly proceed towards the proposed solution, where cloud service QoS values for alternatives, with weights, is passed to the membrane algorithm. This process of weight calculation is done if no satisfactory weights are available according to the user. Thus, the initial portion of IAHP proposed by (Rao, 2007; Saaty, 1983) is used for calculating weights. The final weights used are as in table 5.3. These weights with the data, as in the dataset (Sun et al., 2019), are used for the cloud service selection process (ranking), using ENPS-ITOPSIS and ENPS-MTOPSIS.

Name	Attribute - QoS Parameter	Unit	Beneficial/ Non-Beneficial
C1	Availability	Percentage	Beneficial
C2	Throughput	Invokes / Sec	Beneficial
C3	Successability	Percentage	Beneficial
C4	Reliability	Percentage	Beneficial
C5	Latency	milliseconds	Non-Beneficial
C6	Response Time	milliseconds	Non-Beneficial
C7	Response Time of Customer services	milliseconds	Non-Beneficial
C8	Cost	US Dollar	Non-Beneficial

Table 5.1 QoS Parameters and their details (Sun et al., 2019)

Services	C1	C2	C3	C4	C5	C6	C7	C8
Service 1	94	2.1	98	73	30.84	124.17	112.508	39.0861
Service 2	56	5	58	73	104.975	408.21	17	5.8709

Table 5.2 Sample service parameter values format

Attributes	Weight
C1	0.1935
C2	0.09536
C3	0.116
C4	0.08775
C5	0.10796
C6	0.12861
C7	0.07223
C8	0.1935

Table 5.3 Weights considered for QoS Parameters

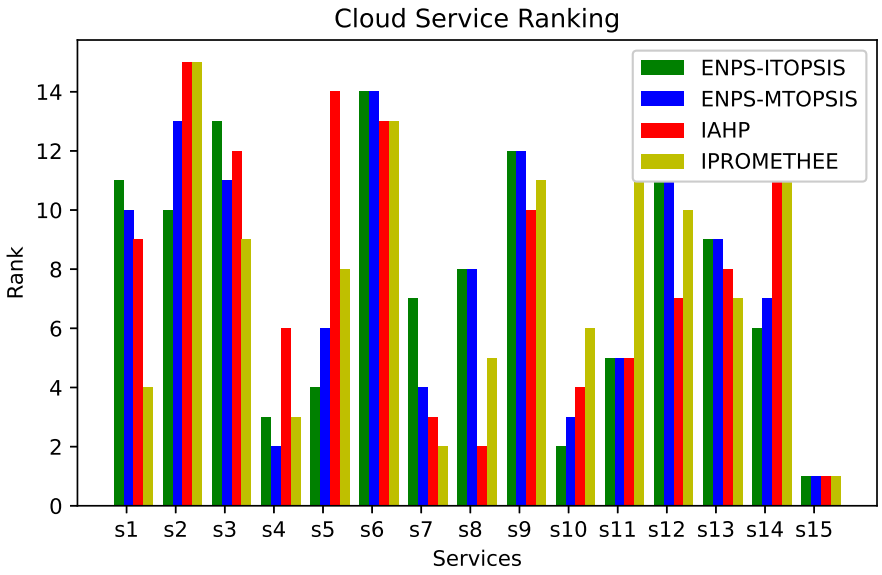

Figure 5.11 Ranks of Services

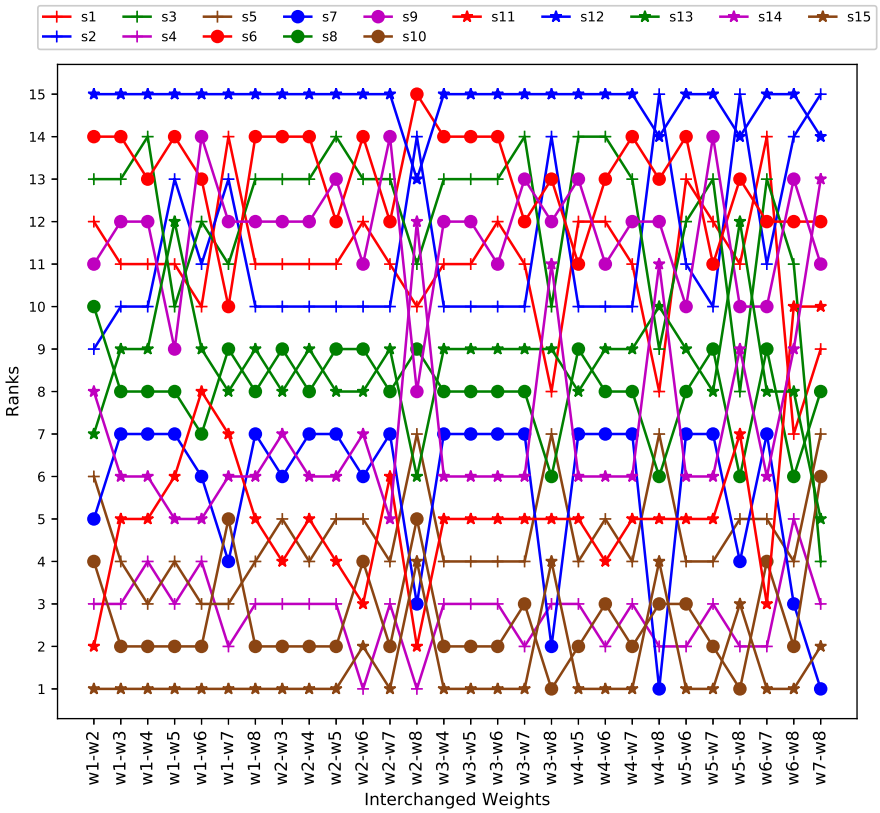

Figure 5.12 Sensitivity Analysis for ENPS-ITOPSIS

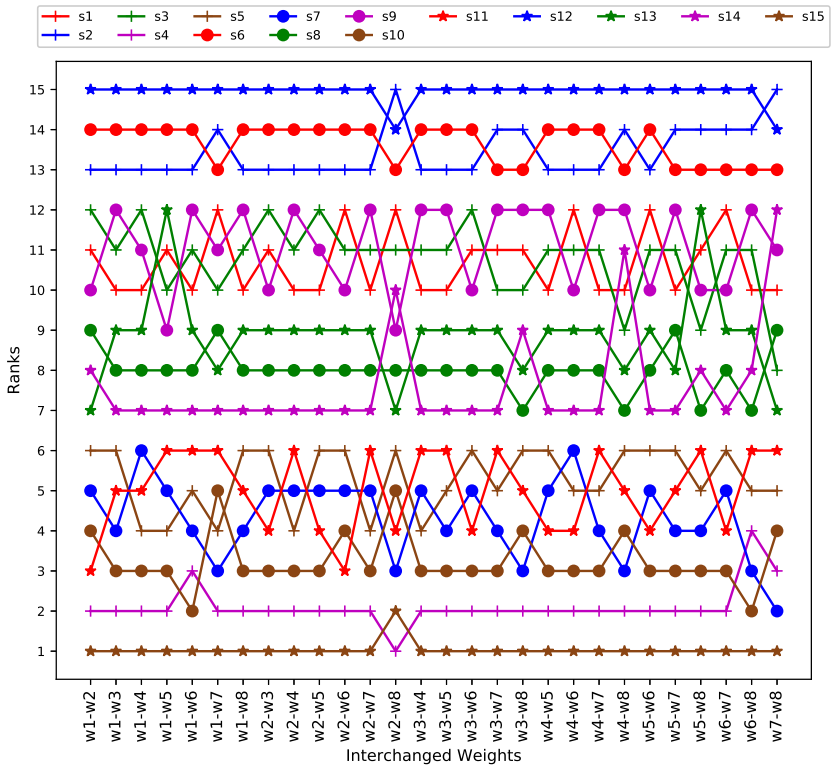

Figure 5.13 Sensitivity Analysis for ENPS-MTOPSIS

5.5 Results and Analysis

As per the data given in the previous section, the corresponding algorithm is executed, after passing the weights and the data of the cloud services, to the membrane system. Accordingly, ranks are obtained for the 15 services ($s1 - s15$) that have been considered with the sample, as given in table 5.2. The ranks obtained for both the approaches (ENPS-ITOPSIS and ENPS-MTOPSIS) are plotted against two popular algorithms, namely IAHP and IPROMETHEE, as in figure 5.11, where the x-axis represent the cloud services and y-axis represent the ranks.

There are two main methods used here for ascertaining the robustness of the proposed models: Sensitivity Analysis and Kendall Tau Distance Rank (KTDR). Determining the robustness is the primary aim of the methods, as the ENPS structure ensures that they are maximally parallel. KTDR independently does not give any useful information about sensitivity, but it is a means to measure the results obtained as part of sensitivity analysis, quantitatively. Thus, the first step involves getting the sensitivity details (weight inter-

changed ranks) with the next step being the use of KTDR, to determine the sensitivity quantitatively.

5.5.1 Sensitivity Analysis

A sensitivity analysis is done on the data of all the considered services ($s1 - s15$) to ascertain the robustness of the proposed approaches. The weights are interchanged pairwise, to check the effect of the change on the resultant ranks. As the weights are interchanged, the ranks are recalculated. If the ranks change for majority of the cases, then the method is said to be sensitive; otherwise, it is said to be robust. There are eight attributes (w1...w8) considered in this problem. When pairwise interchange of weights is done, there are overall 28 possibilities (${}^8C_2 = \frac{8!}{6!2!}$), and all of them are considered.

ENPS-ITOPSIS:

The results obtained for sensitivity analysis for ENPS-ITOPSIS are shown in figure 5.12, where the x-axis shows the weight interchange and y-axis has the ranks. The results show that on several occasions, when weights are interchanged, the ranks also gets affected. An important point observed is that the service with rank one remains almost the same (the least affected), all through the 28 interchanges, with change only in 6 cases, though there are several changes for other services. This can be perceived as moderately sensitive behavior to the changes, though further this is quantitatively proved.

ENPS-MTOPSIS:

The results obtained for sensitivity analysis of ENPS-MTOPSIS are shown in figure 5.13. There are a total of 28 cases of weight swapping considered here. This is similar to the changes considered for ENPS-ITOPSIS method. There is considerably less difference in the reaction, in response to any swap in weights for ENPS-MTOPSIS. Compared to ENPS-ITOPSIS, the change is far less. The changes drastically reduce, and the extreme positions of ranks are almost stable in this case.

Further, the sensitivity of IAHP (as shown in figure 5.14) and IPROMETHEE (as shown in figure 5.15) is also plotted, to have an idea about their robustness, which is

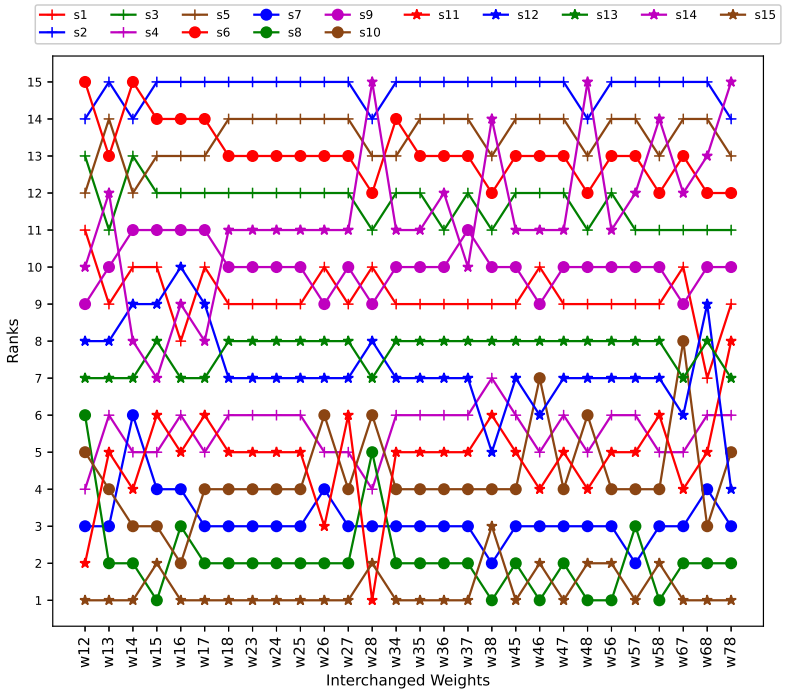

Figure 5.14 Sensitivity Analysis for IAHP

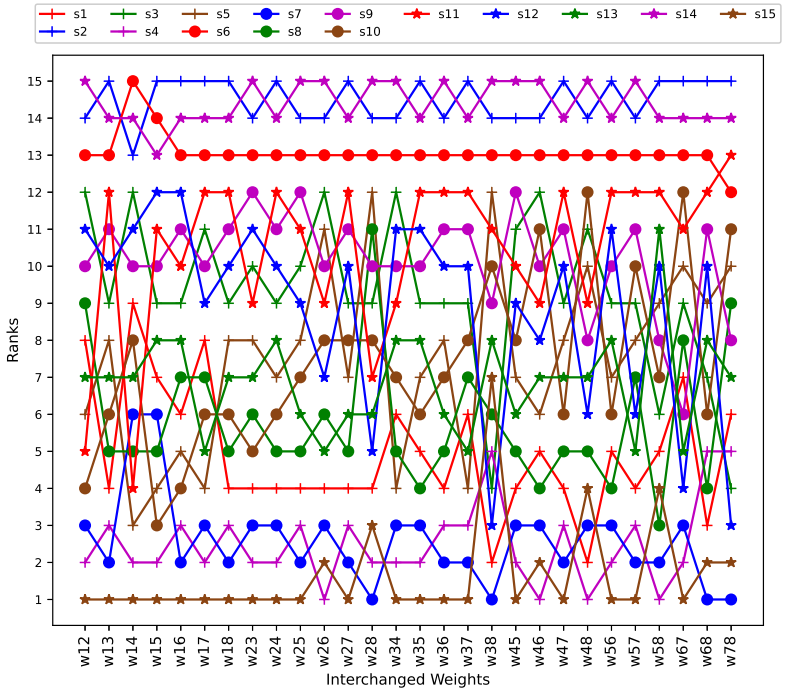

Figure 5.15 Sensitivity Analysis for IPROMETHEE

later quantitatively compared in the next section.

5.5.2 Kendall Tau Distance Ratio

Kendall Tau distance has been proposed by Maurice Kendall (Kendall, 1948). Kendall Tau Distance Ratio (KTDR) is a statistical method used for comparing two ranked lists. It is calculated according to the formula, as given in the equations 5.8, 5.9 and 5.10.

$$\overline{K}_{(i,j)}(L_1,L_2) = \{(L_1(i) < L_1(j) \vee L_2(i) > L_2(j) \wedge L_1(i) > L_1(j) \vee L_2(i) < L_2(j)) \quad (5.8)$$

$$K(L_1,L_2) = \sum_{(i,j)\in P} \overline{K}_{(i,j)}(L_1,L_2) \quad (5.9)$$

$$K_{kdtr} = K(L_1,L_2)/(n*(n-1)) \quad (5.10)$$

where,

n is the length of the array being compared, which is same for both the arrays.

P is the set of un-ordered pairs

$\overline{K}_{(i,j)}(L_1,L_2) = 0$ if L_1 and L_2 are of same order

$\overline{K}_{(i,j)}(L_1,L_2) = 1$ if L_1 and L_2 are of different order

When two ranked lists are compared, the more the KTDR value, the more is the change in the list and the low value indicates less change.

5.5.3 Quantitative Analysis of Sensitivity

The obtained ranks, after interchanging the weights, are considered and Kendall distance Ratio is calculated between the base rank and the other modified ranks obtained for a single method. Thus, a set of KTDR values for each technique ENPS-ITOPSIS, ENPS-

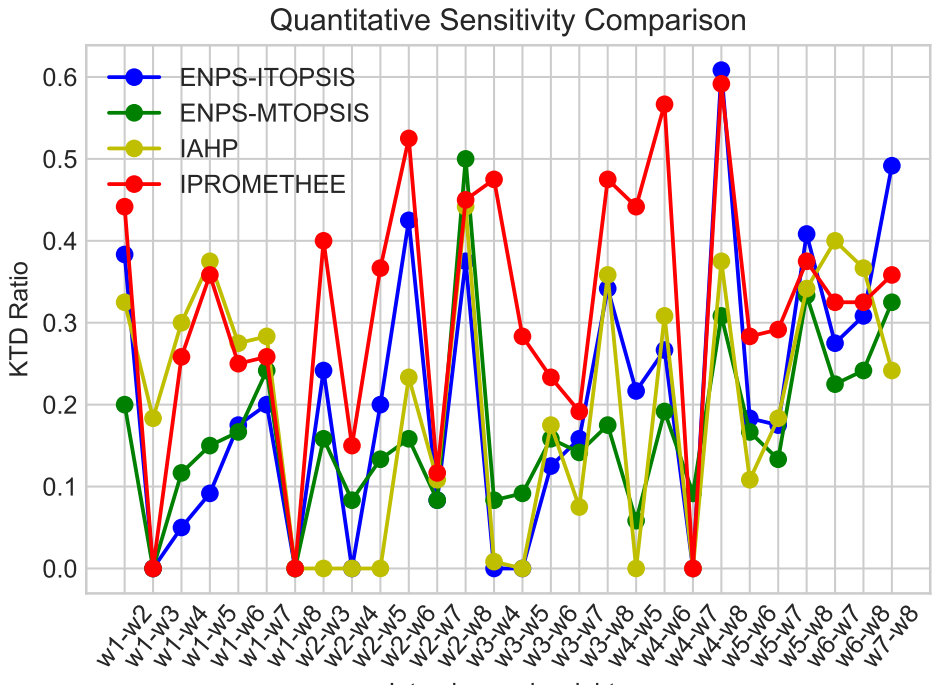

Figure 5.16 Kendall Tau Distance Ratio Comparison

MTOPSIS, IPROMETHEE and IAHP is obtained, as given in figure 5.16, where x-axis is the weight interchange and y axis is the KTDR. Therefore, to put it quantitatively and more precisely, the average KTDR of ENPS-MTOPSIS is 0.168452381, ENPS-ITOPSIS is 0.206547619, IAHP is 0.195238095 and IPROMETHEE is 0.313988095 As per the values and graphs, it is perceived that among the two methods proposed in this work, ENPS-MTOPSIS is the most robust, with the least average KTDR value, among the four methods compared. The ENPS-ITOPSIS method is not as robust as ENPS-MTOPSIS, but it is better than IPROMETHEE. Thus, it can be said that ENPS-ITOPSIS is a moderately sensitive model.

5.6 Summary

Proper Service selection in cloud has long been a challenging problem. There are several methods developed for solving this particular problem. Out of the approaches that have been developed, almost all of them are serial in nature and many of them are sensitive. Here a couple of robust and parallel Bio-inspired membrane-based algorithms for service

selection in cloud are proposed. Membrane computing is an inherently parallel computing paradigm based on a living cell. A membrane algorithm has the capability to be maximally parallel. Here a Multi-criteria Decision Making technique Improved Technique of Order Preference Similarity to the Ideal Solution (ITOPSIS) based membrane algorithms named ENPS-ITOPSIS and it is variant ENPS-MTOPSIS are proposed and implemented. The results are obtained and are analyzed with several statistical techniques and are compared with two standard methods. The results show that ENPS-MTOPSIS is more robust than the others and ENPS-ITOPSIS is better than one of the methods. Thus one method is robust and the other one is moderately sensitive and both of the them are inherently parallel.

Chapter 6

CLOUD WORKFLOW SCHEDULING BASED ON P SYSTEM

The third objective deals with workflow scheduling in cloud. The objective is to create a workflow scheduling algorithm for cloud, thus a parallel workflow scheduling algorithm using membrane computing paradigm is developed. Membrane computing paradigm is realized using Enzymatic Numerical P System (ENPS). The scheduling aspect is realised using Heterogeneous Earliest Finish Time algorithm logic. The proposed algorithm has been implemented using GPUPeP (Simulator for ENPS) and Python. Workflow Scheduling dataset of different number of tasks and workflow structure is used.

6.1 Introduction

Workflows are a set of interdependent tasks that have usually one entry point and one exit point. A workflow is aimed at doing a single big task collectively divided by inter-dependent smaller tasks. A simple workflow structure is given in figure 6.2. Workflow Scheduling in general is a method of mapping tasks to computational resources satisfying the objective functions imposed by users. A workflow contains many tasks which are dependent on each other. It is a NP-Complete problem, and there is no polynomial time algorithm to accurately perform workflow scheduling.

Cloud Workflow Scheduling is a process of assigning the limited stock of Virtual Machines (VMs) to the tasks of the workflows satisfying certain criteria. In this case the

criteria is minimizing the makespan. Makespan is the time taken between starting and completion of the first task of the workflow and last task of the workflow respectively. It is one of the main objective of workflow scheduling.

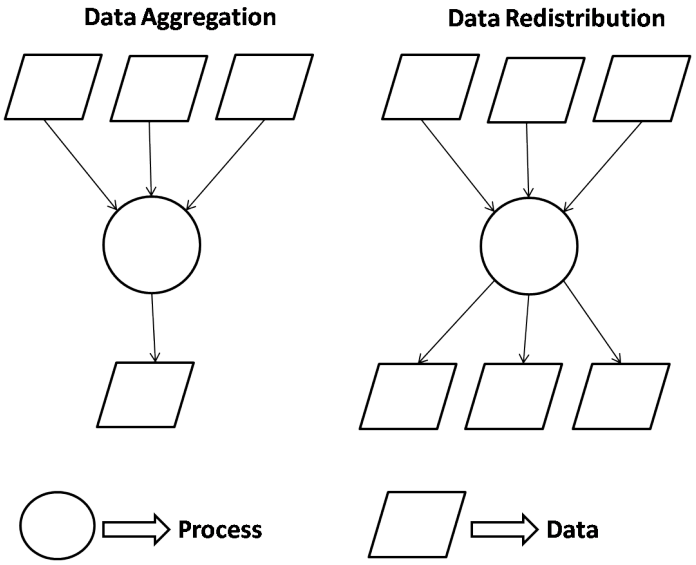

Figure 6.1 Basic Workflow Components (Bharathi et al., 2008)

A simple structure given by Bharathi et al. (2008) is considered to elaborate on the structure of workflows. In figure 6.2, circles represent job or task and parallelogram represents the data (Pegasus, 2019). These kind of similar structure, can be found in most of the workflows. Process is the first and simple structure as in the figure 6.2, which takes input and produces output. The second structure is pipeline, which processes data in a sequential manner (Bharathi et al., 2008). It is a repetition of process structure with multiple data and jobs arranged sequentially. Third structure is data distribution, which is used to partition the data into multiple portions so that other jobs at next level consumes the partitioned data and executes in parallel. But, partitioning data is most time consuming operation in the workflow.

Data aggregation is one of the basic operations in a workflow as shown in figure 6.2. Data aggregation reduces the parallelism of tasks. Sometimes the data is required to be distributed, in this case aggregation proves to be potential bottleneck which may hinder parallelism. Figure 6.2 represents simple workflow in cloud as a directed acyclic graph. In cloud computing, IaaS offers a suitable option for workflow administration with computa-

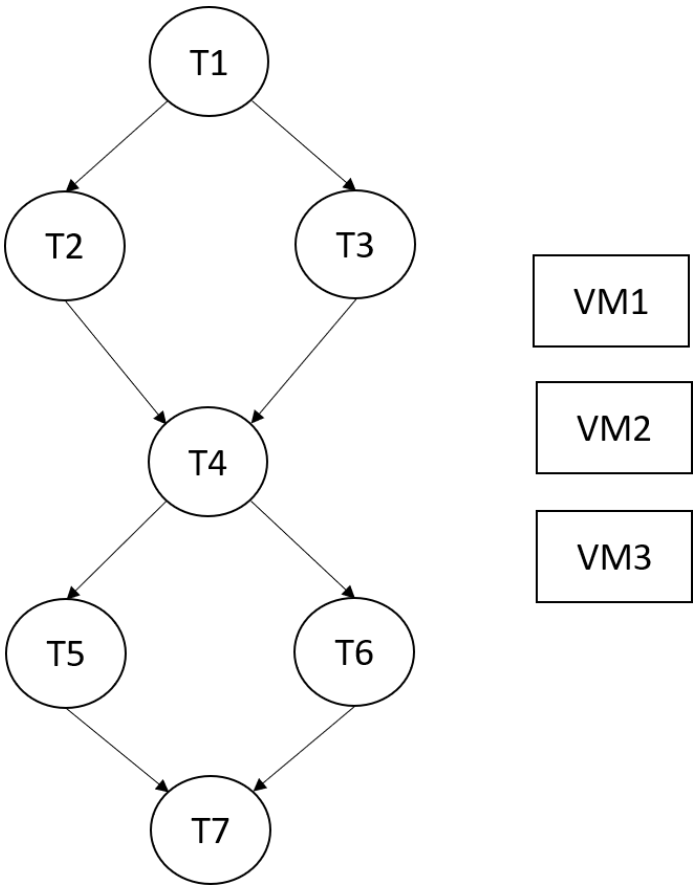

Figure 6.2 Cloud Workflow Basic Structure

tional components. In particular, they give access to VMs of various kinds. These VMs can be used on-request, they can be rented whenever they are required and terminated when they are not. As a rule, clients are charged just for what they utilize, as a rule in addition to charging period characterized by the supplier. This flexibility makes environment of IaaS perfect for the execution of logical workflows. There are several approaches that have been used for workflow scheduling in cloud which are discussed in the literature review (section 2.1), where it is inferred that, there are very less parallel methods for workflow scheduling. Thus, an inherently parallel membrane computing model based on a heuristic approach is proposed for workflow scheduling in cloud.

6.2 Workflow Scheduling in Cloud using P System

Workflow Scheduling in Cloud is a complex process and thus a parallel, P System based model proposed. As per the literature (Section 2.1), there are several P System variants to

choose. As the workflows involve numerical value comparison, Enzymatic Numerical P System (ENPS) is deemed suitable for model. For sequential scheduling logic, Heterogeneous Earliest Finish Time (HEFT) is considered as the base (Topcuoglu et al., 2002).

The P System based method developed in this work primarily involves generating an order with which the tasks are scheduled on heterogeneous components. This process is being done for a cloud based workflow and thus the heterogeneous components are VMs that are of different sizes. To generate rank, two important data about the workflow is required:

- The execution time of each task on the available VM
- The communication cost between each and every VM which transfers control to other VM in the current workflow

It is represented by using an adjacency matrix. The whole process is logically similar to HEFT algorithm proposed by Topcuoglu et al. (2002), but it is structurally and methodically different.

There are primarily two sequential steps that are involved in scheduling a workflow. The logical equivalent of the method is as given in figure 6.3.

- Creating a sequence of tasks based for scheduling
- Scheduling it and calculating the makespan of the actual schedule generated.

The first part of the process involves sequence calculation and the second part is mapping the sequences generated. The first component of the sequence calculation, which is strictly based on ENPS and consists of two independent membrane systems. These independent membrane systems are executed in sequence as is figure 5.6. A membrane generator is developed, which is used to automatically generate membrane system as as part of solution. The membrane systems for sequence calculation is considered and elaborated.

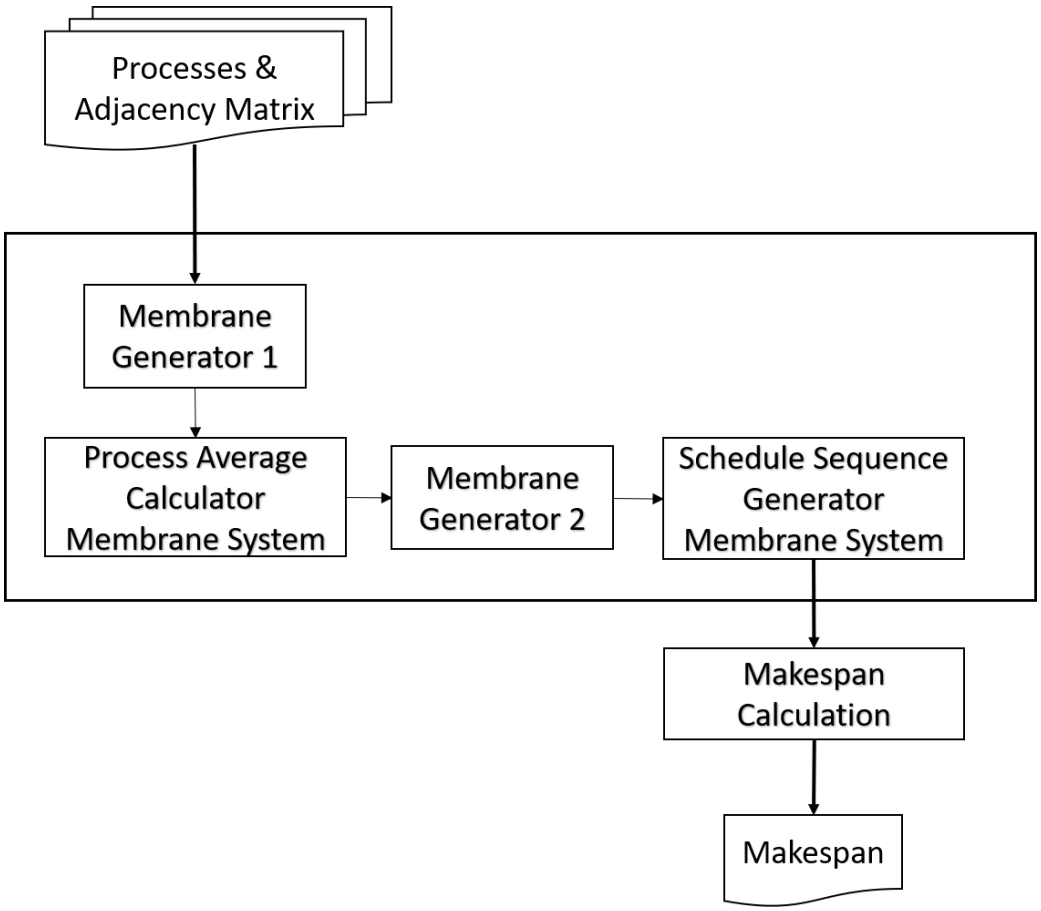

Figure 6.3 P System based Workflow Model

$$\underline{Sub-problem\,1\;-\;M_1}$$

$$\begin{array}{c} a_{1,1}[value],\ldots,a_{n,1}[value] \\ \cdots \\ \cdots \\ z_{1,1}[value],\ldots,z_{n,1}[value],enz_{1,1} \end{array}$$

$$\begin{array}{l} Pr_{1,1}:\ a_{1,1}+\ldots+a_{n,1}\ \ \langle enz_{1,1}\rightarrow\rangle\ \ 1\mid a_{1,1} \\ \cdots \\ Pr_{n,1}:\ a_{1,1}+\ldots+z_{n,1}\ \ \langle enz_{1,1}\rightarrow\rangle\ \ 1\mid a_{n,1} \end{array}$$

Figure 6.4 Membrane System for Sub-problem 1

Membrane System 1:

The first membrane system consists of a single membrane which is designed to calculate the average of the processes available. This average calculated is sent to the next membrane system2 where it is used for finding the values based on which the sequence is

scheduled. The second membrane system is the most important and complex, it involves several programs and is elaborated as follows.

Membrane System 2:

There are several steps for calculating the scheduling sequence. This membrane system is a single membrane system and for any number of tasks the structure remains the same. The first step is to calculate the rank-up function recursively but there is no provision to perform recursion using ENPS. Thus, the whole process is replicated without using recursion. The whole process of execution involves calculating the values in the reverse order of the nodes present i.e. the adjacent matrix is accessed in reverse order from the last row. The enzyme n controls the maximum value operation which involves 10 important programs. Maximum value calculation is a three step process (as in chapter 5).

There are total of a $4n^2+6n$ programs where, n refers to number of tasks considered. There are a total of a $3n$ cycles (steps) used for this process. The total number of enzymes are n^2+2n. The sequential equivalent requires implementation of recursion, which can not be directly used in this case. Rather, in this study the positive use of adjacency matrix is considered as the recursion is realised. The execution starts by activating the last enzyme and subsequently they execute in reverse order. Initially, enzyme number $2n$ is activated. This activates n programs which do pre-processing, related to finding maximum value among the children of current node. After pre-processing, the next step involves calculating the actual maximum value. This process involves n programs with the resultant being passed on to the next programs. The next most important step involves $2n$ programs, but it is also a one step cycle. Two objectives are simultaneously achieved, one is updation of the adjacency matrix values and the next is getting the final values of each task. These set of tasks values are calculated at the end of execution of all programs ($k_{1,1}tok_{n-1}andw_{n,1}$) as the final set of values required.

$\underline{Sub-problem2-M_1}$

$$a_{1,1}[value],\ldots,a_{n,1}[value]$$
$$\ldots$$
$$\ldots$$
$$z_{1,1}[value],\ldots,z_{n,1}[value]$$
$$k_{1,1}[value],\ldots,k_{n,1}[value],w_{1,1}[value],\ldots,w_{n,1}[value]$$
$$min_{1,1}[value],zero_{1,1}[value],final_{1,1}[value],enz_{1,1}[value],\ldots,enz_{2n,1}[value]$$
$$e1_{1,1}[value],\ldots,e1_{n,1}[value]$$
$$\ldots$$
$$\ldots$$
$$en_{1,1}[0],\ldots,en_{n,1}[0]$$

$$Pr_{1,1}:\ 2\times(max_{1,1}-a_{1,1})\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid e1_{2,1}+1\mid k_{1,1}$$
$$\ldots$$
$$Pr_{n,1}:\ 2\times(max_{1,1}-a_{n,1})\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid enz_{n,1}+1\mid k_{1,1}$$
$$Pr_{n+1,1}:\ a_{1,1}\times enz_{2,1}\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid enz_{2,1}$$
$$Pr_{n+2,1}:\ max_{1,1}\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid max_{1,1}$$
$$Pr_{n+3,1}:\ e1_{2,1}*0+e1_{3,1}*0+e1_{4,1}*0-e1_{1,1}\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid k_{1,1}$$
$$Pr_{n+4,1}:\ e1_{1,1}*0-e1_{2,1}+\ldots+e1_{4,1}*0\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid k_{1,1}$$
$$\ldots$$
$$Pr_{2n+2,1}:\ e1_{1,1}*0+e1_{2,1}*0+\ldots-e1_{n,1}\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid k_{1,1}$$
$$Pr_{2n+3,1}:\ a_{1,1}*(e1_{1,1}+\ldots+e1_{n,1})\ \langle enz_{2,1}\rightarrow\rangle\ 1\mid e1_{1,1}+\ldots+1\mid e1_{n,1}$$
$$Pr_{2n+4,1}:\ a_{1,1}+20000\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid enz_{1,1}$$
$$Pr_{2n+5,1}:\ (max_{1,1}-k_{1,1}+w_{1,1})*final_{1,1}/final_{1,1}\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid 1\mid k_{1,1}$$
$$Pr_{2n+2,1}:\ (max_{1,1}-k_{1,1}+w_{1,1}+a_{1,1})*final_{1,1}/final_{1,1}\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid a_{1,1}$$
$$Pr_{2n+7,1}:\ (max_{1,1}-k_{2,1}+w_{1,1})*b_{1,1}/b_{1,1}\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid 1\mid k_{1,1}$$
$$Pr_{2n+8,1}:\ (max_{1,1}-k_{1,1}+w_{1,1}+b_{1,1})*b_{1,1}/b_{1,1}\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid b_{1,1}$$

$$\ldots$$
$$Pr_{4n+3,1}:\ (max_{1,1}-k_{1,1}+w_{1,1})*z_{1,1}/z_{1,1}\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid k_{1,1}$$
$$Pr_{4n+4,1}:\ (max_{1,1}-k_{1,1}+w_{1,1}+z_{1,1})*z_{1,1}/z_{1,1}\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid z_{1,1}$$
$$Pr_{4n+5,1}:\ (max_{1,1}\ \langle e1_{1,1}\rightarrow\rangle\ 1\mid max_{1,1}$$
$$\ldots$$
$$\ldots$$
$$Pr_{4n^2+2n-4,1}:\ 2\times(max_{1,1}-z_{1,1})\ \langle enz_{n,1}\rightarrow\rangle\ 1\mid en_{1,1}+1\mid k_{n,1}$$
$$\ldots$$
$$Pr_{4n^2+3n-5,1}:\ 2\times(max_{1,1}-z_{n,1})\ \langle enz_{n,1}\rightarrow\rangle\ 1\mid en_{n,1}+1\mid k_{n,1}$$
$$Pr_{4n^2+3n-4,1}:\ a_{1,1}\times enz_{n,1}\ \langle enz_{n,1}\rightarrow\rangle\ 1\mid enz_{n,1}+1\mid k_{n,1}$$
$$Pr_{4n^+3n-3,1}:\ max_{1,1}\ \langle enz_{n,1}\rightarrow\rangle\ 1\mid max_{1,1}$$
$$Pr_{4n^+3n-2,1}:\ en_{2,1}*0+en_{3,1}*0+\ldots+en_{n,1}*0-en_{1,1}\ \langle en_{1,1}\rightarrow\rangle\ 1\mid k_{n,1}$$
$$Pr_{4n^+3n-1,1}:\ e1_{1,1}*0-e1_{2,1}+\ldots+e1_{n,1}*0\ \langle en_{2,1}\rightarrow\rangle\ 1\mid k_{1,1}$$
$$\ldots$$
$$Pr_{4n^2+4n-3,1}:\ en_{1,1}*0+en_{2,1}*0+\ldots-en_{n,1}\ \langle en_{n,1}\rightarrow\rangle\ 1\mid k_{1,1}$$
$$Pr_{4n^2+4n-2,1}:\ a_{1,1}*(en_{1,1}+\ldots+en_{n,1})\ \langle en_{1,1}\rightarrow\rangle\ 1\mid en_{1,1}+\ldots+1\mid en_{n,1}$$
$$Pr_{4n^2+4n-1,1}:\ a_{1,1}+20000\ \langle en_{1,1}\rightarrow\rangle\ 1\mid enz_{n,1}$$
$$Pr_{4n^2+4n,1}:\ (max_{1,1}-k_{n,1}+w_{n,1})*a_{n,1}/a_{n,1}\ \langle enz_{n-1,1}\rightarrow\rangle\ 1\mid 1\mid k_{n,1}$$
$$Pr_{4n^2+4n+1,1}:\ (max_{1,1}-k_{n,1}+w_{n,1}+a_{n,1})*a_{n,1}/a_{n,1}\ \langle enz_{n-1,1}\rightarrow\rangle\ 1\mid a_{n,1}$$
$$Pr_{4n^2+4n+2,1}:\ (max_{1,1}-k_{n,1}+w_{n,1})*b_{n,1}/b_{n,1}\ \langle enz_{n-1,1}\rightarrow\rangle\ 1\mid 1\mid k_{n,1}$$
$$Pr_{4n^2+4n+3,1}:\ (max_{1,1}-k_{n,1}+w_{n,1}+b_{n,1})*b_{n,1}/b_{n,1}\ \langle enz_{n-1,1}\rightarrow\rangle\ 1\mid b_{n,1}$$
$$\ldots$$
$$Pr_{4n^2+6n-2,1}:\ (max_{1,1}-k_{n,1}+w_{n,1})*z_{n,1}/z_{n,1}\ \langle enz_{n-1,1}\rightarrow\rangle\ 1\mid k_{1,1}$$
$$Pr_{4n^2+6n-1,1}:\ (max_{1,1}-k_{n,1}+w_{n,1}+b_{n,1})*z_{n,1}/z_{n,1}\ \langle enz_{n-1,1}\rightarrow\rangle\ 1\mid b_{n,1}$$
$$Pr_{4n^2+6n,1}:\ (max_{1,1}\ \langle enz_{n-1,1}\rightarrow\rangle\ 1\mid max_{1,4n^2+6n}$$

Figure 6.5 Membrane System for Sub-problem 2

After the first part is executed, the control is transferred to the second part where the schedule sequence generated is mapped and the final makespan is obtained. The final makespan is the parameter that is considered in the work, based on which further comparison with other algorithms are done.

6.3 Results and Analysis

For implementing membrane based model ENPS-based simulator (GPUPeP) is used as the base. A membrane generator is developed for generating the the main membrane-system for workflow sequence generator. This generator is developed using Python 3.0 and can generate the membrane system for any number of tasks.

After the sequence of execution is calculated, the makespan is obtained with the trailing method. The proposed methods has been compared with two other standard methods, namely, Min-min and Max-min, for scheduling workflow based task. These have been chosen to compare with because these are some of the accepted standard methods.

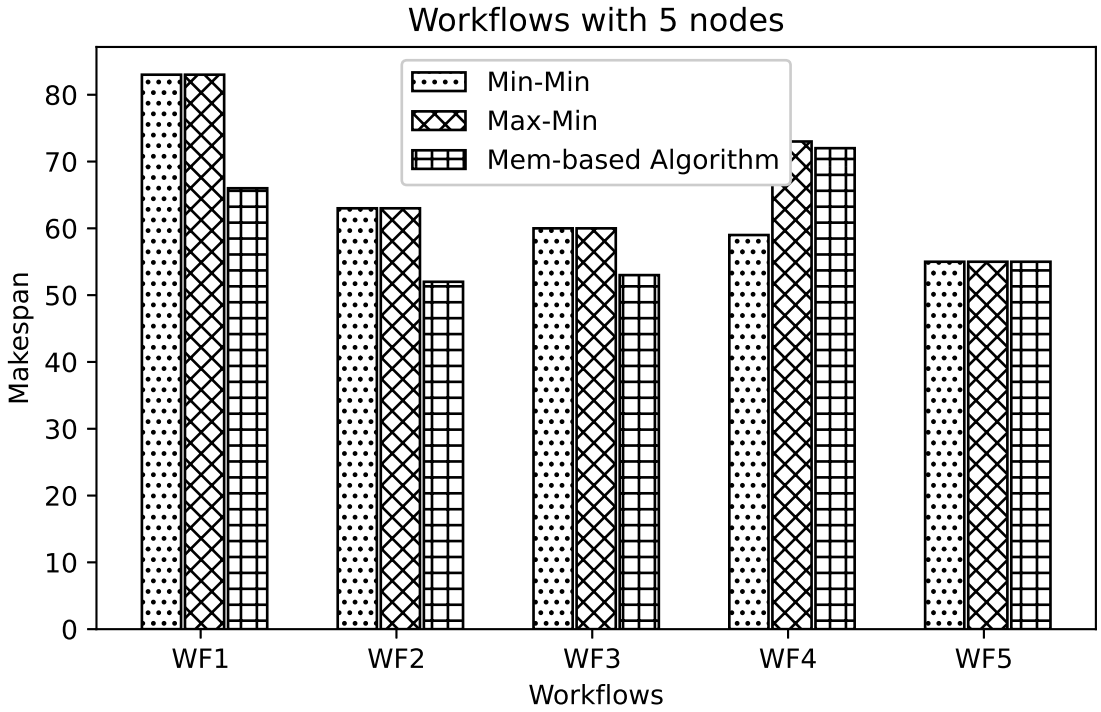

Figure 6.6 Workflow Scheduling for 5 Tasks

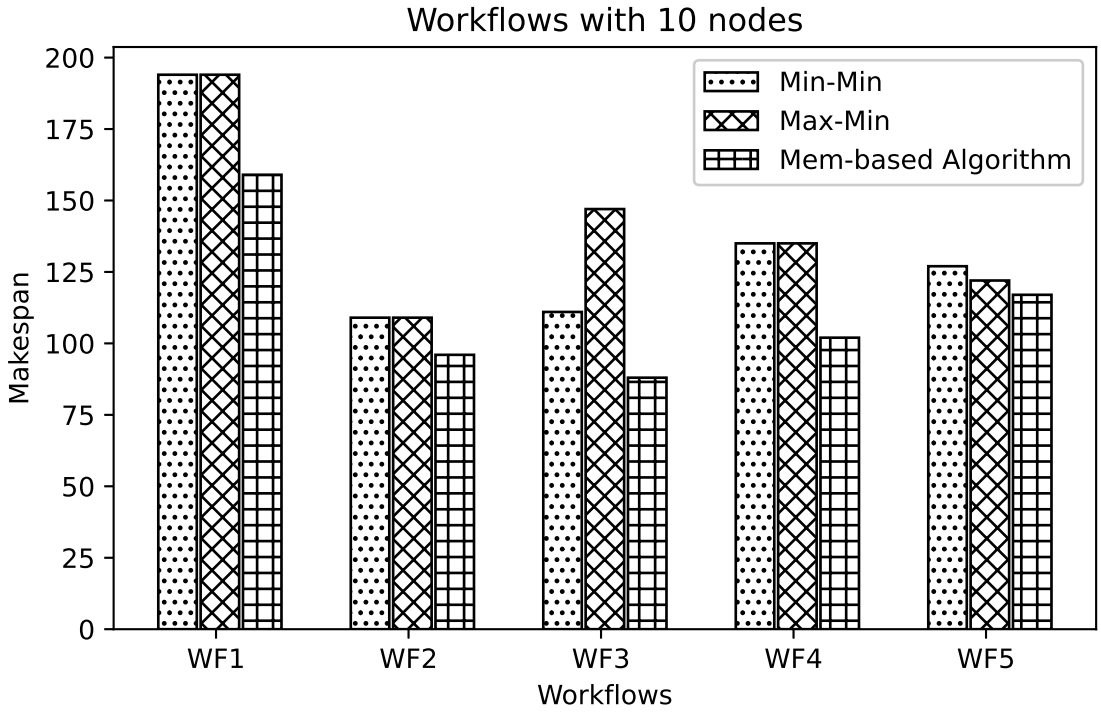

Figure 6.7 Workflow Scheduling for 10 Tasks

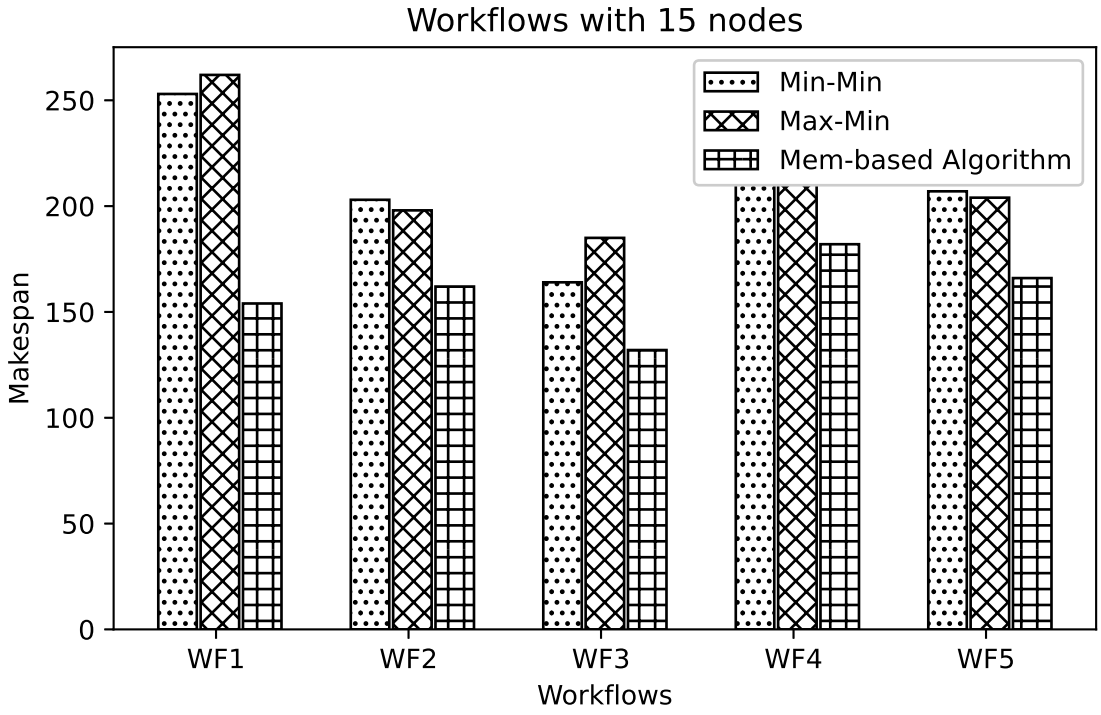

Figure 6.8 Workflow Scheduling for 15 Tasks

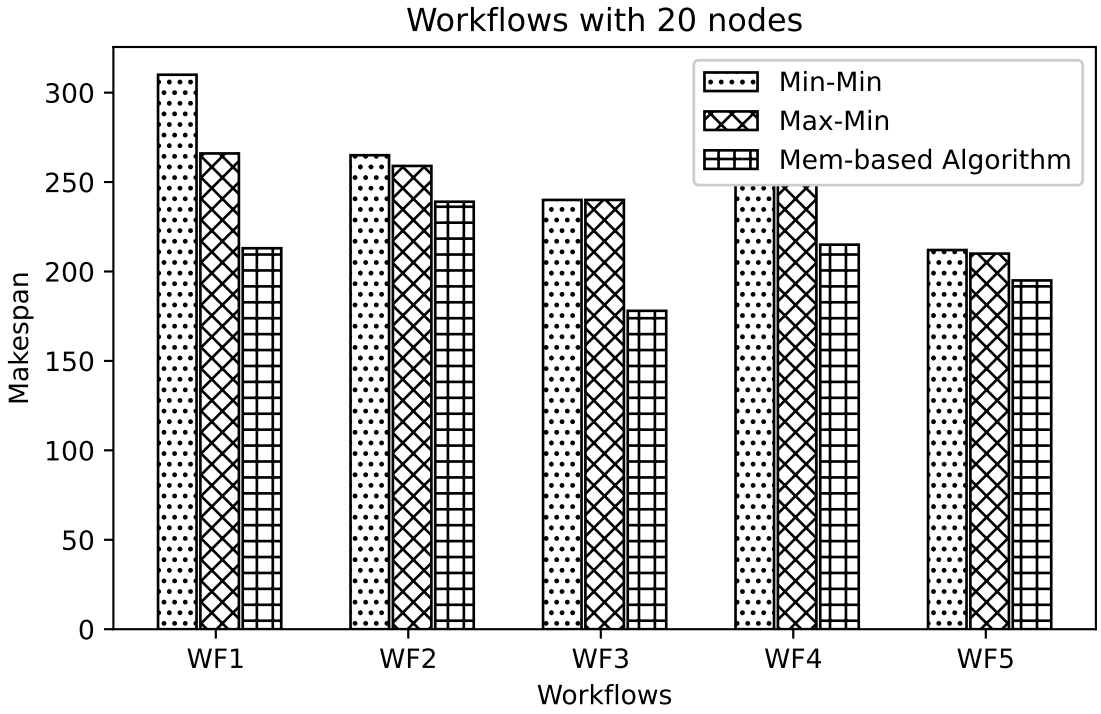

Figure 6.9 Workflow Scheduling for 20 Tasks

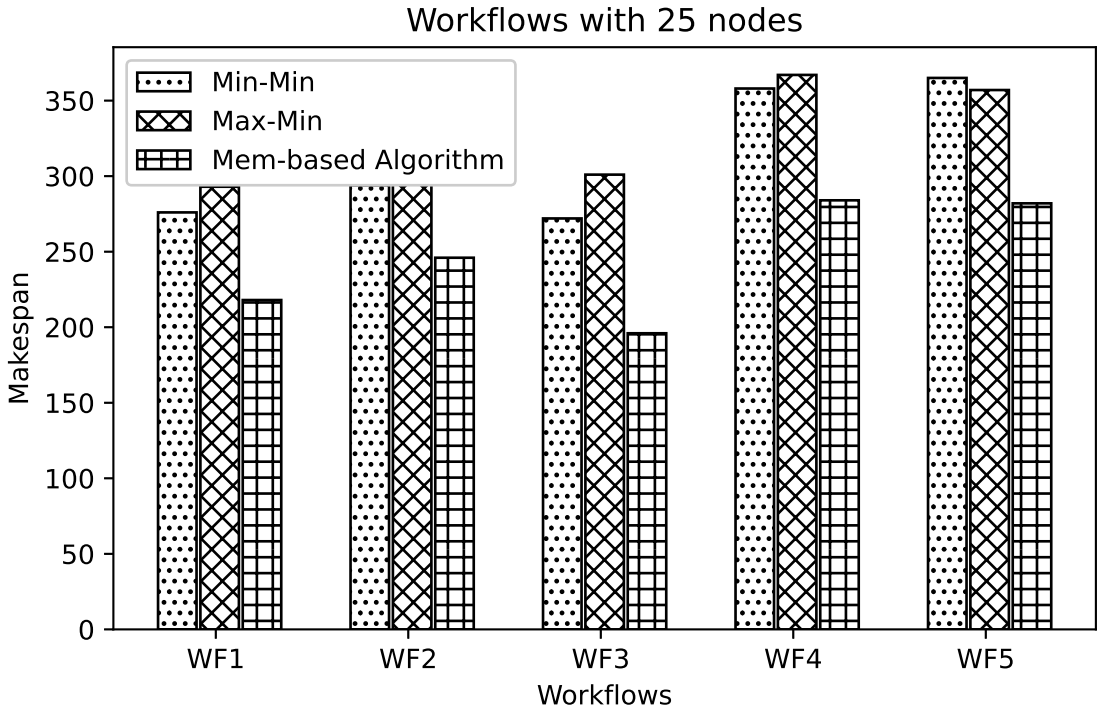

Figure 6.10 Workflow Scheduling for 25 Tasks

Five type of workflows are compared and considered; Five cases each for, five tasks, ten tasks, 15 tasks, 20 tasks and 25 tasks. For each workflow type, five sample workflows of different structures are considered. Figure 6.6 shows results of five different workflows with different graph and different virtual machine configurations are considered with five tasks. The results show that the proposed model is better than other methods on two occasions and it gives same makespan value for the considered cases. This is because the input considered is not so complex and doesn't involve much parallelism (task parallelism). Thus, all the algorithms give similar sequence for few cases.

For next comparison, five different workflows of size 10 are considered. In all the cases the proposed algorithms tends to be better. This can be attributed to the consideration of finish time of algorithm. Similarly when 15 tasks are considered the proposed algorithm fares better than the others compared with as in figure 6.8. Thus few of the cases are significantly better than the others and similar is the case for 20 tasks (figure 6.9).

When 25 tasks are considered to be scheduled for a workflow there are many cases for which the proposed method is better than the other algorithms as in (figure 6.10). Based on the results it can be inferred that the algorithm performs better when the number of tasks increase. Though the improvement is not only based on the number of tasks, it is one of the factor which judges the effectiveness of the algorithm. The backbone of the work is the membrane model, which theoretically has very less time complexity.

6.4 Summary

The proposed algorithm is for scheduling workflows in cloud. The membrane-based algorithm is particularly designed based on ENPS and an efficient heuristic-based solution. Thus, the developed algorithm is ENPS structure-based heuristic algorithm for workflow scheduling in cloud. The proposed multi-membrane system module is designed to be parallel. As part of the study, a workflow membrane generator is also created to automatically create the membranes that represent the workflows. The membranes generated are tested and the results obtained are compared with other standard workflow scheduling methods. Based on the results, the proposed algorithm is found to be better.

Chapter 7

CONCLUSION & FUTURE WORK

The rapid increase in the number of services has increased the complexity of cloud service selection operation, and related to that, there are two primary contribution of this thesis. The first are a couple of cloud service selection algorithms based on P System. The proposed approaches are collectively based on ENPS and MCDM structure. These methods are tested and the results show success as intended, primarily in terms of sensitivity. In addition to this, there are two more approaches using ENPS that have been proposed, one of them is based on outranking method. These works are based on ENPS and are not necessarily designed for cloud based services, but are considered as general MCDM methods. Additionally, two tools which assist the implementation part have been developed. Another important contribution is a workflow based scheduling algorithm based on ENPS and an efficient workflow scheduling heuristic. The proposed approach is tested with several workflows and the results obtained show it to be better than standard methods in terms of makespan.

7.1 Thesis Summary

The thesis begins with an introduction to cloud computing and a formal introduction to service selection for cloud services and workflow scheduling in cloud. It is followed by elaborating literature review and the required background information about important domains and technologies that are used for this thesis. The next chapter deals with service selection, where services are not necessarily cloud services and can be any alternative with

features and associated weights. Two algorithms are proposed (using MCDM and ENPS) and they are tested and the results are obtained. The next chapter is related to tools that have been developed as part of implementation to enhance the results that are primarily aimed at contributing to the ENPS. The next chapter is an important one where a couple of algorithms for cloud service selection are proposed. Both the solutions are established on MCDM based model and are inherently parallel. The results are obtained and are compared with standard algorithms. Chapter 6 deals with workflows in cloud. Particularly, a workflow scheduling problem is solved here. A heuristic-based approach using ENPS is used. The aim here is to reduce the makespan. The proposed approach is tested with several workflows and the results obtained are compared with standard approaches. The results show it to be better than the compared approaches. Chapter 7 presents the summary and conclusion of the research work. This chapter also introduces some possible future expansions to this work.

7.2 Conclusion

Cloud Service Selection is an important process in cloud. As the number of cloud service providers have increased, the process of selecting the best service is becoming complex and there is a need for methods that handle this complexity better. This study proposes an inherently parallel model for service selection in cloud. The proposed model is based on inherently parallel, natural computing paradigm, with Multi-criteria Decision Making as the base. There are two approaches (ENPS-ITOPSIS and ENPS-MTOPSIS) specifically designed for cloud service selection, which are based on Enzymatic Numerical P System (ENPS). These are tested and the results have been elaborated which ascertain the robustness and parallelism of the methods. Further two more methods are proposed (ENPS-IAHP and ENPS-IPROMETHEE), these can be used as general MCDM methods for general service selection (item selection) problem. The implementation subsequently has lead to designing two important tools for bolstering the implementation. A GPU-based simulator and a multi-membrane system executor are designed, which are used as facilitators for implementing service selection algorithms. Another important contribution of this thesis is an approach for workflow scheduling in cloud. This is one of the complex

problems that have been considered. This problem is solved, using heuristic and ENPS. The approach is tested on several workflows and the results are compared with two standard approaches. The proposed method is inherently parallel and found to be better than other approaches in terms of makespan. Thus, all the proposed approaches are inherently parallel and are better in some aspect according to the considered problem.

7.3 Future Work

A couple of future extensions are envisioned and identified during the course of this research work.

- As future work, an enhancement can be made in the area of service selection in cloud; a mechanism which is better than the current MCDM mechanism can be considered as the base. Another work that can be considered here is using any other membrane-based approach for service selection in cloud i.e. a different combination of sequential approach and membrane model.

- Further, a specific type of membrane-based model for NP-Complete problem can be designed, which can be a generic solution for all the similar problems as workflow scheduling.

Bibliography

Abdel-Basset, M., Mohamed, M., and Chang, V. (2018). NMCDA: A framework for evaluating cloud computing services. *Future Generation Computer Systems*, 86, 12–29.

Abrishami, S. and Naghibzadeh, M. (2012). Deadline-constrained workflow scheduling in Software as a Service cloud. *Scientia Iranica*, 19(3), 680–689.

Abrishami, S., Naghibzadeh, M., and Epema, D. H. (2013). Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1), 158–169.

Adhikari, M. and Amgoth, T. (2016). Efficient algorithm for workflow scheduling in cloud computing environment. In *2016 Ninth International Conference on Contemporary Computing (IC3)*, 1–6. IEEE.

Adhikari, M. and Amgoth, T. (2019). An intelligent water drops-based workflow scheduling for IaaS cloud. *Applied Soft Computing*, 77, 547–566.

Ahmed, T., Verma, R., Bakshi, M., and Srivastava, A. (2014). Membrane Computing Inspired Approach for Executing Scientific Workflow in the Cloud. In *International Conference on Membrane Computing*, 51–65. Springer.

Al-Faifi, A., Song, B., Hassan, M. M., Alamri, A., and Gumaei, A. (2019). A hybrid multi-criteria decision method for cloud service selection from Smart data. *Future Generation Computer Systems*, 93, 43–57.

Al-Faifi, A. M., Song, B., Hassan, M. M., Alamri, A., and Gumaei, A. (2018). Data on performance prediction for cloud service selection. *Data in brief*, 20, 1039–1043.

Arabnejad, V., Bubendorfer, K., and Ng, B. (2018). Budget and deadline aware e-science workflow scheduling in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 30(1), 29–44.

Ardelean, I. I. and Cavaliere, M. (2003). Modelling biological processes by using a probabilistic P System software. *Natural Computing*, 2(2), 173–197.

Arroyo, F., Luengo, C., Baranda, A. V., and de Mingo, L. (2002). A software simulation of transition P Systems in Haskell. In *Workshop on Membrane Computing*, 19–32. Springer.

Askarnejad, S., Malekimajd, M., and Movaghar, A. (2018). Network and Application-Aware Cloud Service Selection in Peer-Assisted Environments. *IEEE Transactions on Cloud Computing*.

Bangalan, Z. F., Soriano, K. A. N., Juayong, R. A. B., Cabarle, F. G. C., Adorna, H. N., and Martínez del Amor, M. Á. (2013). A GPU simulation for evolution-communication P Systems with energy having no antiport rules. *Proceedings of the Eleventh Brainstorming Week on Membrane Computing, 25-50. Sevilla, ETS de Ingeniería Informática, 4-8 de Febrero, 2013,*.

Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., and Vahi, K. (2008). Characterization of scientific workflows. In *2008 third workshop on workflows in support of large-scale science*, 1–10. IEEE.

Bianco, L., Manca, V., Marchetti, L., and Petterlini, M. (2007). Psim: a simulator for biomolecular dynamics based on P Systems. In *2007 IEEE Congress on Evolutionary Computation*, 883–887. IEEE.

Blakes, J., Twycross, J., Romero, F. J., Krasnogor, N., et al. (2011). The Infobiotics Workbench: an integrated in silico modelling platform for Systems and Synthetic Biology. *Bioinformatics*, 27(23), 3323–3324.

Bonchiş, C., Ciobanu, G., Izbaşa, C., and Petcu, D. (2005). A Web-based P Systems simulator and its parallelization. In *International Conference on Unconventional Computation*, 58–69. Springer.

Borrego-Ropero, R., Dıaz-Pernil, D., and Pérez-Jiménez, M. J. (2007). Tissue simulator: A graphical tool for tissue P Systems. In *Proceedings of the international workshop automata for cellular and molecular computing. Satellite of the 16th international symposium on fundamentals of computational theory. MTA SZTAKI, Budapest, Hungary*, 23–34.

Brandusa Pavel, A. (2011). Membrane controllers for cognitive robots. *Department of Automatic Control and System Engineering, Politehnica University of Bucharest, Romania, Master's Thesis*.

Brans, J.-P. (1982). *L'ingénierie de la décision: l'élaboration d'instruments d'aide a la décision*. Université Laval, Faculté des sciences de l'administration.

Buiu, C., Arsene, O., Cipu, C., and Patrascu, M. (2011). A Software tool for Modeling and Simulation of Numerical P Systems. *BioSystems*, 103(3), 442–447.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer Systems*, 25(6), 599–616.

Byun, E.-K., Kee, Y.-S., Kim, J.-S., and Maeng, S. (2011). Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8), 1011–1026.

Cabarle, F., Adorna, H., and Martinez-del Amor, M. A. (2011a). Simulating spiking neural p systems without delays using gpus. *International Journal of Natural Computing Research (IJNCR)*, 2(2), 19–31.

Cabarle, F. G. C., Adorna, H., and Martínez, M. A. (2011b). A Spiking Neural P System simulator based on CUDA. In *International Conference on Membrane Computing*, 87–103. Springer.

Cabarle, F. G. C., Adorna, H., and Martinez-del Amor, M. A. (2011c). An improved GPU simulator for spiking neural P Systems. In *Sixth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2011*, 262–267. IEEE.

Cabarle, F. G. C., de la Cruz, R. T. A., Cailipan, D. P. P., Zhang, D., Liu, X., and Zeng, X. (2019). On solutions and representations of spiking neural P Systems with rules on synapses. *Information Sciences*, 501, 30–49.

Calheiros, R. N. and Buyya, R. (2013). Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Transactions on Parallel and Distributed Systems*, 25(7), 1787–1796.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1), 23–50.

Campos, M., Llorens, C., Sempere, J. M., Futami, R., Rodriguez, I., Carrasco, P., Capilla, R., Latorre, A., Coque, T. M., Moya, A., et al. (2015). A membrane computing simulator of trans-hierarchical Antibiotic Resistance Evolution dynamics in nested ecological compartments (ARES). *Biology direct*, 10(1), 1.

Casas, I., Taheri, J., Ranjan, R., Wang, L., and Zomaya, A. Y. (2017). A balanced scheduler with data reuse and replication for scientific workflows in cloud computing Systems. *Future Generation Computer Systems*, 74, 168–178.

Casas, I., Taheri, J., Ranjan, R., Wang, L., and Zomaya, A. Y. (2018). GA-ETI: An enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. *Journal of computational science*, 26, 318–331.

Castellini, A. and Manca, V. (2008). MetaPlab: a computational framework for metabolic P Systems. In *International Workshop on Membrane Computing*, 157–168. Springer.

Cecilia, J. M., García, J. M., Guerrero, G. D., Martínez-del Amor, M. A., Pérez-Hurtado, I., and Pérez-Jiménez, M. J. (2010a). Simulating a P System based efficient solution to SAT by using GPUs. *The Journal of Logic and Algebraic Programming*, 79(6), 317–325.

Cecilia, J. M., García, J. M., Guerrero, G. D., Martínez-del Amor, M. A., Pérez-Hurtado, I., and Pérez-Jiménez, M. J. (2010b). Simulation of P Systems with active membranes on CUDA. *Briefings in Bioinformatics*, 11(3), 313–322.

Chen, H., Zhu, X., Qiu, D., Liu, L., and Du, Z. (2017a). Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds. *IEEE Transactions on Parallel and distributed Systems*, 28(9), 2674–2688.

Chen, W., Xie, G., Li, R., Bai, Y., Fan, C., and Li, K. (2017b). Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing Systems. *Future Generation Computer Systems*, 74, 1–11.

Ciobanu, G. and Paraschiv, D. (2002). P System software simulator. *Fundamenta Informaticae*, 49(1-3), 61–66.

Ciobanu, G., Păun, G., and Pérez-Jiménez, M. J. (2006). *Applications of membrane computing*, 17. Springer.

Ciobanu, G., Paun, G., and Stefanescu, G. (2003). Sevilla carpets associated with P Systems. In *Proceedings of the Brainstorming Week on Membrane Computing, Tarragona, Spain*, 135–140.

Ciobanu, G. and Wenyuan, G. (2003). P Systems running on a cluster of computers. In *International Workshop on Membrane Computing*, 123–139. Springer.

Cook, S. (2012). *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes.

Cordón-Franco, A., Gutiérrez-Naranjo, M. A., Pérez-Jiménez, M. J., and Sancho-Caparrini, F. (2004). A PROLOG simulator for deterministic P Systems with active membranes. *New Generation Computing*, 22(4), 349–363.

Das, D. K. and Renz, T. (2006). A Simulation Model for P Systems with Active Membranes. In *2006 IEEE Conference on Emerging Technologies-Nanoelectronics*, 338–340. IEEE.

Dassow, J. and Păun, G. (1999). On the power of membrane computing. *Journal of Universal Computer Science*, 5(2), 33–49.

Dehghan-Manshadi, B., Mahmudi, H., Abedian, A., and Mahmudi, R. (2007). A novel method for materials selection in mechanical design: Combination of non-linear normalization and a modified digital logic method. *Materials & Design*, 28(1), 8–15.

Deng, H., Yeh, C.-H., and Willis, R. J. (2000). Inter-company comparison using modified TOPSIS with objective weights. *Computers & Operations Research*, 27(10), 963–973.

Díaz-Pernil, D., Fernández-Mírquez, C. M., García-Quismondo, M., Gutiérrez-Naranjo, M. A., and Martínez-del Amor, M. A. (2010). Solving Sudoku with Membrane Computing. In *2010 IEEE fifth international conference on bio-inspired computing: theories and applications (BIC-TA)*, 610–615. IEEE.

Díaz Pernil, D., Pérez Hurtado de Mendoza, I., Pérez Jiménez, M. d. J., and Riscos Núñez, A. (2008). P-lingua: A programming language for membrane computing. *Proceedings of the Sixth Brainstorming Week on Membrane Computing, 135-155. Sevilla, ETS de Ingeniería Informática, 4-8 de Febrero, 2008*.

Elaziz, M. A., Xiong, S., Jayasena, K., and Li, L. (2019). Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. *Knowledge-Based Systems*, 169, 39–52.

Equinix (Accessed July, 2019). Link: https://www.equinix.com/interconnection-services/cloud-exchange-fabric/.

Eusuff, M., Lansey, K., and Pasha, F. (2006). Shuffled Frog-leaping algorithm: A Memetic meta-heuristic for discrete optimization. *Engineering optimization*, 38(2), 129–154.

Farag, M. M. (2007). *Materials and process selection for engineering design*. CRC Press.

Florea, A. G. and Buiu, C. (2015). Lulu-A software Simulator for P colonies. Use case scenarios and demonstration videos.

Florea, A. G. and Buiu, C. (2016). Development of a Software Simulator for P colonies–Applications in Robotics. *International Journal of Unconventional Computing*, 12.

Florea, A. G. and Buiu, C. (2017). Modelling multi-robot interactions using a generic controller based on numerical P Systems and ROS. In *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 1–6. IEEE.

Florea, A. G. and Buiu, C. (2018). PeP (Enzymatic) Numerical P System simulator. `http://membranecomputing.net/pep/index.html`. [Online; Accessed 30-July-2018].

Frisco, P. and Gibson, R. T. (2005). A Simulator and an Evolution Program for Conformon-P Systems. In *SYNASC*, 7, 26–27.

G. Ciobanu, D. Paraschiv (2001). Membrane software. A P System simulator. Technical report, Pre-proceeedings Workshop on Membrane Computing, Curtea de Arges, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain.

Garcia-Quismondo, M. MeCoGUI. `Link:https://www.p-lingua.org/wiki/index.php/MeCoGUI`.

Garcıa-Quismondo, M. (2013). A Java-Based P-Lingua Simulator for Enzymatic Numerical P Systems. `https://www.cs.us.es/blogs/mgarcia/research/software_tools/java_simulator_enps/`. [Online; Accessed 23-July-2017].

García Quismondo, M., Brandusa Pavel, A., and Pérez Jiménez, M. d. J. (2012a). Simulating large-scale ENPS models by means of GPU. *Proceedings of the Tenth Brainstorming Week on Membrane Computing, 137-152. Sevilla, ETS de Ingeniería Informática, 30 de Enero-3 de Febrero, 2012*.

García Quismondo, M., Brandusa Pavel, A., Pérez Jiménez, M. d. J., et al. (2012b). Simulating Large-Scale ENPS Models by Means of GPU.

García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M. J., and Riscos-Núñez, A. (2009). An overview of P-Lingua 2.0. In *International Workshop on Membrane Computing*, 264–288. Springer.

García-Quismondo, M., Macías-Ramos, L. F., and Pérez-Jiménez, M. J. (2013). Implementing Enzymatic Numerical P Systems for AI applications by means of Graphic Processing Units. In *Beyond Artificial Intelligence*, 137–159. Springer.

García-Quismondo, M., Martínez-del Amor, M. A., and Pérez-Jiménez, M. J. (2014). Probabilistic Guarded P Systems, a new formal modelling framework. In *International Conference on Membrane Computing*, 194–214. Springer.

Garg, S. K., Versteeg, S., and Buyya, R. (2011). SMICloud: A framework for comparing and ranking cloud services. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, 210–218. IEEE.

Gheorghe, M. (2010). System Modeling Framework. Link: http://www.dcs.shef.ac.uk/people/M.Gheorghe/PSimulatorWeb/Tools.htm.

Ghosh, N., Ghosh, S. K., and Das, S. K. (2014). SelCSP: A framework to facilitate selection of cloud service providers. *IEEE transactions on cloud computing*, 3(1), 66–79.

Gutiérrez Naranjo, M. Á., Pérez Jiménez, M. d. J., and Riscos Núñez, A. (2005). A simulator for confluent P systems. *Proceedings of the Third Brainstorming Week on Membrane Computing, 169-184. Sevilla, ETS de Ingeniería Informática, 31 de Enero-4 de Febrero, 2005,*.

Hu, J., Chen, G., Peng, H., Wang, J., Huang, X., and Luo, X. (2017). A kNN classifier optimized by P Systems. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 432–437. IEEE.

Huang, L. and Wang, N. (2006). An optimization algorithm inspired by membrane computing. In *International Conference on Natural Computation*, 49–52. Springer.

Hwang, C.-L. and Yoon, K. (1981). *Multiple Attribute Decision Making*, 186 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, Heidelberg.

Kari, L. and Rozenberg, G. (2008). The many facets of natural computing. *Communications of the ACM*, 51(10), 72–83.

Kaveri, B. A., Gireesha, O., Somu, N., Raman, M. G., and Sriram, V. S. (2017). E-FPROMETHEE: An Entropy based Fuzzy multi criteria decision making service ranking approach for cloud service selection. In *International Conference on Intelligent Information Technologies*, 224–238. Springer.

Kendall, M. G. (1948). *Rank correlation methods*. Griffin.

Kumar, M. and Sharma, S. (2018). PSO-COGENT: Cost and energy efficient scheduling in cloud environment with deadline constraint. *Sustainable Computing: Informatics and Systems*, 19, 147–164.

Kumar, R. R., Mishra, S., and Kumar, C. (2017). Prioritizing the solution of cloud service selection using integrated MCDM methods under Fuzzy environment. *The Journal of Supercomputing*, 73(11), 4652–4682.

Lang, M., Wiesche, M., and Krcmar, H. (2018). Criteria for Selecting Cloud Service Providers: A Delphi Study of Quality-of-Service Attributes. *Information & Management*, 55(6), 746–758.

Lee, S. and Seo, K.-K. (2016). A hybrid multi-criteria decision-making model for a cloud service selection problem using BSC, fuzzy Delphi method and fuzzy AHP. *Wireless Personal Communications*, 86(1), 57–75.

Leporati, A., Porreca, A. E., Zandron, C., and Mauri, G. (2013). Improving universality results on parallel Enzymatic Numerical P Systems. *Proceedings of the Eleventh Brainstorming Week on Membrane Computing, 177-200. Sevilla, ETS de Ingeniería Informática, 4-8 de Febrero, 2013*.

Li, Z., Ge, J., Hu, H., Song, W., Hu, H., and Luo, B. (2015). Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds. *IEEE Transactions on Services Computing*, 11(4), 713–726.

Liang, Y.-C., Chen, A. H.-L., and Nien, Y.-H. (2014). Artificial Bee Colony for workflow scheduling. In *2014 IEEE congress on evolutionary computation (CEC)*, 558–564. IEEE.

Lin, D., Squicciarini, A. C., Dondapati, V. N., and Sundareswaran, S. (2016). A cloud brokerage architecture for efficient cloud service selection. *IEEE Transactions on Services Computing*, 12(1), 144–157.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., and Leaf, D. (2011). NIST Cloud Computing Reference Architecture. *NIST Special Publication*, 500, 292.

Llorente Rivera, D. and Gutiérrez Naranjo, M. Á. (2015). The Pole Balancing Problem with Enzymatic Numerical P Systems. In *Proceedings of the Thirteenth Brainstorming Week on Membrane Computing, 195-206. Sevilla, ETS de Ingeniería Informática, 2-6 de Febrero, 2015*. Fénix Editora.

Lopes, A., Valentim, N., Moraes, B., Zilse, R., and Conte, T. (2018). Applying user-centered techniques to analyze and design a mobile application. *Journal of Software Engineering Research and Development*, 6(1), 1–23.

Ma, H., Hu, Z., Li, K., and Zhang, H. (2016). Toward trustworthy cloud service selection: A time-aware approach using interval neutrosophic set. *Journal of Parallel and Distributed Computing*, 96, 75–94.

Macías-Ramos, L. F., Pérez-Hurtado, I., García-Quismondo, M., Valencia-Cabrera, L., Pérez-Jiménez, M. J., and Riscos-Núñez, A. (2011). A P–Lingua based simulator for spiking neural P Systems. In *International Conference on Membrane Computing*, 257–281. Springer.

Macías-Ramos, L. F., Valencia-Cabrera, L., Song, B., Song, T., Pan, L., and Pérez-Jiménez, M. J. (2015). A P_Lingua Based Simulator for P Systems with Symport/Antiport Rules. *Fundamenta Informaticae*, 139(2), 211–227.

Maciej, M. (2012). Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press*.

Malita, M. (2000). Membrane computing in PROLOG. In *Pre-Proceedings of The Workshop on Multiset Processing (WMP-CdeA 2000)*, page 8.

Mansouri, N., Zade, B. M. H., and Javidi, M. M. (2019). Hybrid Task Scheduling Strategy for Cloud Computing by Modified Particle Swarm Optimization and Fuzzy Theory. *Computers & Industrial Engineering*, 130, 597–633.

Mao, M. and Humphrey, M. (2011). Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–12. IEEE.

Maroosi, A. and Muniyandi, R. C. (2013). Accelerated simulation of membrane computing to solve the n-queens problem on multi-core. In *International Conference on Swarm, Evolutionary, and Memetic Computing*, 257–267. Springer.

Maroosi, A., Muniyandi, R. C., Sundararajan, E., and Zin, A. M. (2014). Parallel and distributed computing models on a graphics processing unit to accelerate simulation of membrane Systems. *Simulation Modelling Practice and Theory*, 47, 60–78.

Maroosi, A., Muniyandi, R. C., Sundararajan, E. A., and Zin, A. M. (2013). Improved implementation of simulation for membrane computing on the graphic processing unit. *Procedia Technology*, 11, 184–190.

Martínez-del Amor, M. A., García-Quismondo, M., Macías-Ramos, L. F., Valencia-Cabrera, L., Riscos-Núñez, A., and Pérez-Jiménez, M. J. (2015). Simulating P Systems on GPU devices: a survey. *Fundamenta Informaticae*, 136(3), 269–284.

Martínez del Amor, M. Á., Karlin, I., Jensen, R. E., Pérez Jiménez, M. d. J., Elster, A. C., et al. (2012). Parallel simulation of probabilistic P Systems on multicore platforms. *Proceedings of the Tenth Brainstorming Week on Membrane Computing,(2) 17-26. Sevilla, ETS de Ingeniería Informática, 30 de Enero-3 de Febrero, 2012*.

Martínez-del Amor, M. A., Macías-Ramos, L. F., Valencia-Cabrera, L., Riscos-Núñez, A., and Pérez-Jiménez, M. J. (2014). Accelerated simulation of P Systems on the GPU: a survey. In *Bio-Inspired Computing-Theories and Applications*, 308–312. Springer.

Martínez del Amor, M. Á., Pérez Carrasco, J., Pérez Jiménez, M. d. J., et al. (2013). Simulating a Family of Tissue P Systems Solving SAT on the GPU. *Proceedings of*

the Eleventh Brainstorming Week on Membrane Computing, 201-220. Sevilla, ETS de Ingeniería Informática, 4-8 de Febrero, 2013,.

Martínez-del Amor, M. A., Pérez-Hurtado, I., García-Quismondo, M., Macías-Ramos, L. F., Valencia-Cabrera, L., Romero-Jiménez, Á., Graciani, C., Riscos-Núñez, A., Colomer, M. A., and Pérez-Jiménez, M. J. (2012). DCBA: Simulating population dynamics P Systems with proportional object distribution. In *International Conference on Membrane Computing*, 257–276. Springer.

Martínez-del Amor, M. A., Pérez-Hurtado, I., Pérez-Jiménez, M. J., and Riscos-Núñez, A. (2010). A P-Lingua based simulator for tissue P Systems. *The Journal of Logic and Algebraic Programming*, 79(6), 374–382.

Martínez-Puras, A. and Pacheco, J. (2016). MOAMP-Tabu search and NSGA-II for a real Bi-objective scheduling-routing problem. *Knowledge-Based Systems*, 112, 92–104.

Mell, P., Grance, T., et al. (2011). The NIST definition of cloud computing.

Mohammadi, A., Asadi, H., Mohamed, S., Nelson, K., and Nahavandi, S. (2018). Optimizing model predictive control horizons using genetic algorithm for motion cueing algorithm. *Expert Systems with Applications*, 92, 73–81.

Nasonov, D., Butakov, N., Balakhontseva, M., Knyazkov, K., and Boukhanovsky, A. V. (2014). Hybrid evolutionary workflow scheduling algorithm for dynamic heterogeneous distributed computational environment. In *International Joint Conference SOCO'14-CISIS'14-ICEUTE'14*, 83–92. Springer.

Nawaz, F., Asadabadi, M. R., Janjua, N. K., Hussain, O. K., Chang, E., and Saberi, M. (2018). An MCDM method for cloud service selection using a Markov chain and the best-worst method. *Knowledge-Based Systems*, 159, 120–131.

Nepomuceno Chamorro, I. d. l. Á. (2004). A Java simulator for basic transition P systems. *Proceedings of the Second Brainstorming Week on Membrane Computing, 309-315. Sevilla, ETS de Ingeniería Informática, 2-7 de Febrero, 2004*.

Nepomuceno Chamorro, I. d. l. Á., Nepomuceno Chamorro, J. A., Romero Campero, F. J., et al. (2005). A tool for using the SBML format to represent P Systems which model biological reaction networks.

Nishida, T. Y. (2006). Membrane algorithms: approximate algorithms for NP-complete optimization problems. In *Applications of membrane computing*, 303–314. Springer.

Noval, D. B., Jiménez, M. J. P., and Caparrini, F. S. (2002). A MzScheme implementation of transition P Systems. In *Workshop on Membrane Computing*, 58–73. Springer.

Nvidia (2019). CUDA C++ Programming Guide. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`. [Online; Accessed 30-April-2019].

Orellana Martín, D., Graciani Díaz, C., Martínez del Amor, M. Á., Riscos Núñez, A., Valencia Cabrera, L., et al. (2014). Revisiting Sevilla Carpets: A New Tool for the P-Lingua Era. *Proceedings of the Twelfth Brainstorming Week on Membrane Computing, 281-292. Sevilla, ETS de Ingeniería Informática, 3-7 de Febrero, 2014,.*

Pandey, S., Wu, L., Guru, S. M., and Buyya, R. (2010). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *2010 24th IEEE international conference on advanced information networking and applications*, 400–407. IEEE.

Patiniotakis, I., Verginadis, Y., and Mentzas, G. (2015). PuLSaR: Preference-based Cloud Service Selection for cloud service brokers. *Journal of Internet Services and Applications*, 6(1), 26.

Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences*, 61(1), 108–143.

Paun, G. (2012). *Membrane computing: An Introduction*. Springer Science & Business Media.

Păun, G. and Păun, R. (2006). Membrane computing and economics: Numerical P Systems. *Fundamenta Informaticae*, 73(1, 2), 213–227.

Păun, G. and Rozenberg, G. (2002). A guide to Membrane Computing. *Theoretical Computer Science*, 287(1), 73–100.

Paun, G., Rozenberg, G., and Salomaa, A. (2010). *The Oxford Handbook of Membrane Computing*. IOS Press.

Pavel, A. B., Arsene, O., and Buiu, C. (2010). Enzymatic Numerical P Systems-A new class of membrane computing Systems. In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, 1331–1336. IEEE.

Pavel, A. B. and Buiu, C. (2012). Using Enzymatic Numerical P Systems for modeling mobile robot controllers. *Natural Computing*, 11(3), 387–393.

Pavel, A. B., Vasile, C. I., and Dumitrache, I. (2012). Robot localization implemented with Enzymatic Numerical P Systems. In *Conference on Biomimetic and Biohybrid Systems*, 204–215. Springer.

Pegasus (2019). Pegasus - Workflow Gallery. `http://arxiv.org/abs/1707.07435`. [Online] Last accessed: 20-10-2019.

Peng, H., Jiang, Y., Wang, J., and Pérez-Jiménez, M. (2015). Membrane clustering algorithm with hybrid evolutionary mechanisms. *Journal Software*, 26(5), 1001–1012.

Peng, H., Jin, J., and Wang, J. (2016). Parallel Implementation of Membrane Computing-Inspired Clustering Algorithm on Graphics Processing Unit. *Journal of Computational and Theoretical Nanoscience*, 13(6), 3673–3680.

Peng, H., Shao, J., Li, B., Wang, J., Pérez Jiménez, M. d. J., Jiang, Y., and Yang, Y. (2012). Image thresholding with cell-like P Systems. *Proceedings of the Tenth Brainstorming Week on Membrane Computing,(2) 75-88. Sevilla, ETS de Ingeniería Informática, 30 de Enero-3 de Febrero, 2012,.*

Peng, H., Shi, P., Wang, J., Riscos-Núñez, A., and Pérez-Jiménez, M. J. (2017). Multiobjective fuzzy clustering approach based on tissue-like membrane Systems. *Knowledge-Based Systems*, 125, 74–82.

Peng, H., Wang, J., Pérez-Jiménez, M. J., and Shi, P. (2013). A novel image thresholding method based on membrane computing and fuzzy entropy. *Journal of Intelligent & Fuzzy Systems*, 24(2), 229–237.

Peng, H., Zhang, J., Jiang, Y., Huang, X., and Wang, J. (2014). DE-MC: A membrane clustering algorithm based on differential evolution mechanism. *Romanian Journal Information Science and Technology*, 17(1), 76–88.

Pérez-Hurtado, I., Martínez-del Amor, M. Á., Zhang, G., Neri, F., and Pérez-Jiménez, M. J. A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. *Integrated Computer-Aided Engineering*, (Preprint), 1–18.

Pérez-Hurtado, I., Pérez-Jiménez, M. J., Zhang, G., and Orellana-Martín, D. (2018). Robot path planning using rapidly-exploring random trees: A membrane computing approach. In *2018 7th International Conference on Computers Communications and Control (ICCCC)*, 37–46. IEEE.

Perez-Hurtado, I., Valencia-Cabrera, L., Chacon, J. M., Riscos-Nunez, A., and Perez-Jimenez, M. J. (2014). A P–Lingua based Simulator for Tissue P Systems with Cell Separation. *Sceince and Technology*, 17(1), 89–102.

Pérez-Hurtado, I., Valencia-Cabrera, L., Pérez-Jiménez, M. J., Colomer, M. A., and Riscos-Núñez, A. (2010). MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, 637–643. IEEE.

Pérez Jiménez, M. d. J. and Romero Campero, F. J. (2004). A CLIPS simulator for recognizer P Systems with active membranes. *Proceedings of the Second Brainstorming Week on Membrane Computing, 387-413. Sevilla, ETS de Ingeniería Informática, 2-7 de Febrero, 2004*.

Pescini, D., Besozzi, D., Mauri, G., and Zandron, C. (2006). Dynamical probabilistic P Systems. *International Journal of Foundations of Computer Science*, 17(01), 183–204.

Ramírez Martínez, D. and Gutiérrez Naranjo, M. Á. (2007). A software tool for dealing with spiking neural P systems. *Proceedings of the Fifth Brainstorming Week on Membrane Computing, 299-313. Sevilla, ETS de Ingeniería Informática, 29 de Enero-2 de Febrero, 2007*.

Rao, R. V. (2007). *Decision making in the manufacturing environment: using graph theory and fuzzy multiple attribute decision making methods*. Springer Science & Business Media.

Ravie, C. and Ali, M. (2015). Enhancing the Simulation of Membrane System on the GPU for the N-Queens Problem. *Chinese Journal of Electronics*, 24(4), 740–743.

Read, J. (2014). CloudHarmony. com. *Cloud Server Performance Benchmarking*.

RGNC. PMCGPU. `Link:http://www.p-lingua.org/wiki/index.php/PMCGPU`.

RGNC (2016). Software - The P Systems Page. `Link:http://ppage.pSystems.eu/index.php/Software`.

Rimal, B. P. and Maier, M. (2016). Workflow scheduling in multi-tenant cloud computing environments. *IEEE Transactions on parallel and distributed Systems*, 28(1), 290–304.

Rivero-Gil, E., Gutiérrez-Naranjo, M. A., Romero-Jiménez, Á., and Riscos-Núñez, A. (2011). A software tool for generating graphics by means of P Systems. *Natural Computing*, 10(2), 879–890.

Rodriguez, M. A. and Buyya, R. (2014). Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE transactions on cloud computing*, 2(2), 222–235.

Saaty, T. (1983). Decision making for leaders : the Analytic Hierarchy Process for decisions in a complex world. *The Engineering Economist*, 29(1), 74–75.

Saaty, T. L. (1988). What is the Analytic Hierarchy Process? In *Mathematical models for decision support*, 109–121. Springer.

Sedwards, S. and Mazza, T. (2007). Cyto-Sim: a formal language model and stochastic simulator of membrane-enclosed biochemical processes. *Bioinformatics*, 23(20), 2800–2802.

Siegel, J. and Perdue, J. (2012). Cloud services measures for global use: the Service Measurement Index (SMI). In *2012 Annual SRII Global Conference*, 411–415. IEEE.

Singh, G. and Deep, K. (2016). A new membrane algorithm using the rules of Particle Swarm Optimization incorporated within the framework of cell-like P-Systems to solve Sudoku. *Applied Soft Computing*, 45, 27–39.

Singh, G., Deep, K., and Nagar, A. K. (2014). Cell-like P-Systems based on rules of Particle Swarm Optimization. *Applied Mathematics and Computation*, 246, 546–560.

Soltani, S., Martin, P., and Elgazzar, K. (2018). A hybrid approach to automatic IaaS service selection. *Journal of Cloud Computing*, 7(1), 12.

Somu, N., Kirthivasan, K., and Sriram, V. S. (2017). A rough set-based hypergraph trust measure parameter selection technique for cloud service selection. *The Journal of Supercomputing*, 73(10), 4535–4559.

Somu, N., MR, G. R., Kirthivasan, K., and VS, S. S. (2018). A trust-centric optimal service ranking approach for cloud service selection. *Future Generation Computer Systems*, 86, 234–252.

Su, S., Li, J., Huang, Q., Huang, X., Shuang, K., and Wang, J. (2013). Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39(4-5), 177–188.

Sun, L., Dong, H., Hussain, F. K., Hussain, O. K., and Chang, E. (2014). Cloud service selection: State-of-the-art and future research directions. *Journal of Network and Computer Applications*, 45, 134–150.

Sun, L., Dong, H., Hussain, O. K., Hussain, F. K., and Liu, A. X. (2019). A framework of cloud service selection with criteria interactions. *Future Generation Computer Systems*, 94, 749–764.

Sun, L., Ma, J., Zhang, Y., Dong, H., and Hussain, F. K. (2016). Cloud-FuSeR: Fuzzy ontology and MCDM based cloud service selection. *Future Generation Computer Systems*, 57, 42–55.

Sundareswaran, S., Squicciarini, A., and Lin, D. (2012). A brokerage-based approach for cloud service selection. In *2012 IEEE Fifth International Conference on Cloud Computing*, 558–565. IEEE.

Suzuki, Y. and Tanaka, H. (2000). On a LISP implementation of a class of P Systems. *Romanian Journal of Information Science and Technology*, 3(2), 173–186.

Syropoulos, A., Mamatas, E. G., Allilomes, P. C., and Sotiriades, K. T. (2003). A distributed simulation of transition P Systems. In *International Workshop on Membrane Computing*, 357–368. Springer.

Topcuoglu, H., Hariri, S., and Wu, M.-y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed Systems*, 13(3), 260–274.

Ur Rehman, Z., Hussain, O. K., and Hussain, F. K. (2014). Parallel cloud service selection and ranking based on QoS history. *International Journal of Parallel Programming*, 42(5), 820–852.

Vaidya, O. S. and Kumar, S. (2006). Analytic hierarchy process: An overview of applications. *European Journal of Operational Research*, 169(1), 1–29.

Verma, A. and Kaushal, S. (2017). A hybrid multi-objective Particle Swarm Optimization for scientific workflow scheduling. *Parallel Computing*, 62, 1–19.

Wang, H., Peng, H., Shao, J., and Wang, T. (2012). A thresholding method based on P Systems for image segmentation. *ICIC Express Letters*, 6(1), 221–227.

Wang, X., Zhang, G., Gou, X., Paul, P., Neri, F., Rong, H., Yang, Q., and Zhang, H. Multi-behaviors coordination controller design with enzymatic numerical P Systems for robots. *Integrated Computer-Aided Engineering*, (Preprint), 1–22.

Wang, X., Zhang, G., Zhao, J., Rong, H., Ipate, F., and Lefticaru, R. (2015). A modified membrane-inspired algorithm based on particle swarm optimization for mobile robot path planning. *International Journal of Computers Communications & Control*, 10(5), 732–745.

Wu, C. Q., Lin, X., Yu, D., Xu, W., and Li, L. (2014). End-to-end delay minimization for scientific workflows in clouds under budget constraint. *IEEE Transactions on Cloud Computing*, 3(2), 169–181.

Wu, Q., Ishikawa, F., Zhu, Q., Xia, Y., and Wen, J. (2017). Deadline-constrained cost optimization approaches for workflow scheduling in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 28(12), 3401–3412.

Wu, Z., Ni, Z., Gu, L., and Liu, X. (2010). A revised discrete particle swarm optimization for cloud workflow scheduling. In *2010 International Conference on Computational Intelligence and Security*, 184–188. IEEE.

Yadav, N. and Goraya, M. S. (2018). Two-way ranking based service mapping in cloud environment. *Future Generation Computer Systems*, 81, 53–66.

Yao, G., Ding, Y., and Hao, K. (2017). Using imbalance characteristic for fault-tolerant workflow scheduling in cloud Systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(12), 3671–3683.

Zhang, F., Cao, J., Hwang, K., Li, K., and Khan, S. U. (2014a). Adaptive workflow scheduling on cloud computing platforms with iterativeordinal optimization. *IEEE Transactions on Cloud Computing*, 3(2), 156–168.

Zhang, G., Gheorghe, M., Pan, L., and Perez-Jimenez, M. J. (2014b). Evolutionary membrane computing: a comprehensive survey and new results. *Information Sciences*, 279, 528–551.

Zhang, G., Pérez-Jiménez, M. J., and Gheorghe, M. (2017a). Robot Control with Membrane Systems. In *Real-life Applications with Membrane Computing*, 213–258. Springer.

Zhang, H., Peng, Y., Tian, G., Wang, D., and Xie, P. (2017b). Green material selection for sustainability: A hybrid MCDM approach. *PLOS ONE*, 12(5), e0177578.

Zhu, Z., Zhang, G., Li, M., and Liu, X. (2015). Evolutionary multi-objective workflow scheduling in cloud. *IEEE Transactions on parallel and distributed Systems*, 27(5), 1344–1357.

LIST OF PUBLICATIONS/ CONFERENCE PAPERS

Journal Publications

[1] S. Raghavan, Shanthanu S. Rai, M. P. Rohit, and K. Chandrasekaran. "GPUPeP: Parallel Enzymatic Numerical P System simulator with a Python-based interface. "*Biosystems, Elsevier* (2020): 104186. [SCI] [IF: 1.8].

[2] S. Raghavan, Yashas Gangadhar, Varun Pattar, and K. Chandrasekaran. "Multi-ENPS simulator support tool with automatic file inter-conversion and multi-membrane execution.". *Biosystems, Elsevier* 189 (2020): 104067. [SCI] [IF: 1.8].

[3] S. Raghavan and K. Chandrasekaran (2021). "Membrane-based Models for Service Selection in Cloud", *Information Sciences, Elsevier*, 558, 103-123. [SCIE] [IF: 5.9].

[4] S. Raghavan and K. Chandrasekaran. “ENPS-IPROMETHEE: Enzymatic Numerical P System based Improved Preference Ranking Organization Method for Enrichment Evaluation”. (Communicated)

[5] S. Raghavan and K. Chandrasekaran. “Enzymatic Numerical P System based Workflow Scheduling in Cloud”. (Communicated)

Conference Proceedings

[1] S. Raghavan, K Chandrasekaran," Enzymatic Numerical P System for Improved Analytic Hierarchy Process", 6^{th} *Asian Conference on Membrane Computing 2017 (ACMC 2017)*, Chengdu, China, 2017.

[2] S. Raghavan, and K. Chandrasekaran. "Tools and simulators for membrane computing-a literature review."*International Conference on Bio-Inspired Computing: Theories and Applications*, pp. 249-277. Springer, Singapore, 2016.

[3] S. Raghavan, and K. Chandrasekaran. "Analysis of emerging workflow scheduling algorithms in cloud." *International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pp. 81-87. IEEE, 2015.

BIO-DATA

Name : Santhanam Raghavan

Email Id : raghavan.sm2005@gmail.com

Date of Birth : 24-11-1990

Address : G8, Anand Square Apartments,

30/36, Govindan Road,

West Mambalam,

Chennai - 600033.

Educational Qualifications:

Degree	Year of Passing	University
B.Tech.	2012	SASTRA University, Thanjavur.
M.Tech	2015	National Institute of Technology Karnataka Surathkal.