CAPITAL UNIVERSITY OF SCIENCE AND
TECHNOLOGY, ISLAMABAD

Task Scheduling Optimization in Cloud Computing

by

Rohail Gulbaz

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2020

I dedicate my dissertation to my beloved family, who supported me throughout my academic life. Their endless encouragement and love empowered me to pursue academic achievements.

CERTIFICATE OF APPROVAL

Task Scheduling Optimization in Cloud Computing

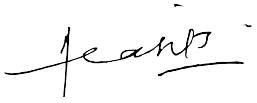
by

Rohail Gulbaz

(MCS181027)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Ayyaz Hussain	QAU, Islamabad
(b)	Internal Examiner	Dr. Aamer Nadeem	CUST, Islamabad
(c)	Supervisor	Dr. Abdul Basit Siddiqui	CUST, Islamabad



Dr. Abdul Basit Siddiqui
Thesis Supervisor
May, 2020

Dr. Nayyer Masood
Head
Dept. of Computer Science
May, 2020

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
May, 2020

Author's Declaration

I, **Rohail Gulbaz** hereby state that my MS thesis titled "**Task Scheduling Optimization in Cloud Computing**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

Rohail Gulbaz
Registration No: MCS181027

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**Task Scheduling Optimization in Cloud Computing**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been dully acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

Rohail Gulbaz
Registration No: MCS181027

Acknowledgements

I praise and worship my God almighty who is all in all. His love endures forever and every achievement and blessing in my life belong to Him. He is a perfect source of strength in my life.

I want to show gratitude to my family including parents and sisters for their endless support and love. I would not have achieved anything without them. I also want to thank all my teachers who have taught and guided me. They have a special role in enlightening me with knowledge.

Specially, I want to thank my supervisor **Dr. Abdul Basit Siddiqui** for this thesis work and his valuable contribution in guiding and assisting me. I also acknowledge that I have learnt a lot during the time I have spent in this university.

Rohail Gulbaz
Registration No: MCS181027

Abstract

Cloud computing is immensely adopted in the modern age and cloud users are significantly increasing. Due to the substantial demand of cloud services the provision of quality of services is becoming more challenging. Cloud systems utilize the virtualization to efficiently scale the physical resources to provide uninterrupted and seamless services. Cloud users interact with the cloud system and their requests are considered as jobs and necessary actions are taken to respond them. The jobs are becoming more intensive and these require computationally intensive processing. The resources in cloud systems are heterogeneous and different scheduling heuristics are opted to map jobs to available resources. The cloud service providers endeavor to improve the resource utilization and making the jobs execution efficient which ultimately benefit their system and the customer satisfaction increases. In the literature, the metrics of better makespan and maximum resource utilization are observed to be fundamental for the satisfaction of cloud service providers and consumers.

Different heuristics and meta-heuristics have been successfully applied to schedule independent heterogeneous jobs to heterogeneous Virtual Machines (VMs). However, the meta-heuristics possess the potential to explore large search space of possible solutions unlike the heuristics that haste to a rapid conclusion. Makespan and resource utilization being two different objectives are conflicting in nature, where one is minimization and other is maximization function. The multi-objective optimization is essential to find better solution regarding makespan and resource utilization. Evolutionary genetic approach is flexible enough for defining balance in exploration and exploitation. In search of the optimal solution, the genetic approach undergoes slow convergence. The BGA is presented which fuses the balancer mechanism in the GA, so that the makespan can be improved while not disturbing the balance of workload among the resources. The SGA is presented to improve the convergence of GA and it is applicable when fast discovery of optimal solution is required.

The performance is evaluated based on makespan and ARUR where ARUR is a measure to compute the resource utilization. The presented techniques are assessed with state-of-the-art techniques on synthetic and realistic datasets. The BGA shows significant improvement in makespan and resource utilization over DSOS, MGGS, ETA-GA and RALBA. The SGA shows faster convergence as compared to DSOS and ETA-GA by finding a better solution in less number of iterations.

Contents

Author's Declaration iv

Plagiarism Undertaking v

Acknowledgements vi

Abstract vii

List of Figures xi

List of Tables xiii

Abbreviations xiv

Symbols xv

1 Introduction 1
- 1.1 Heterogeneous Distributed Computing . . . 3
- 1.2 Scheduling Issues . . . 4
- 1.3 Taxonomy of Tasks . . . 6
- 1.4 Scheduling Techniques . . . 7
- 1.5 Evolutionary Algorithms . . . 9
- 1.6 CloudSim . . . 9
- 1.7 Problem Statement . . . 10
- 1.8 Research Questions . . . 10
- 1.9 Purpose . . . 11
- 1.10 Scope . . . 11
- 1.11 Significance of Solution . . . 12
- 1.12 Dissertation Organization . . . 12

2 Literature Review 13
- 2.1 Heuristics . . . 13
- 2.2 Meta-Heuristics . . . 17
- 2.3 Research Gap Analysis . . . 33

3 Methodology **34**
3.1 Multi-objective Optimization . . . 35
3.2 Scheduling Approach . . . 38
3.3 Genetic Algorithm . . . 42
3.3.1 Problem Encoding . . . 42
3.3.2 Initialization . . . 43
3.3.3 Fitness Function . . . 44
3.3.4 Selection Operation . . . 52
3.3.5 Crossover . . . 54
3.3.6 Mutation . . . 57
3.4 BGA . . . 61
3.4.1 Balancer Algorithm . . . 63
3.4.2 Parameters Setting of BGA . . . 65
3.5 SGA . . . 66
3.5.1 SOS . . . 67
3.5.2 Parameters Setting of SGA . . . 72
3.6 Application and Analysis . . . 72
3.7 Performance Model and Metrics . . . 74
3.7.1 Makespan . . . 74
3.7.2 ARUR . . . 75
3.7.3 Convergence . . . 75
3.8 Evaluation Benchmarks . . . 75
3.9 Dataset . . . 78
3.9.1 Synthetic Dataset . . . 79
3.9.2 Realistic Dataset . . . 80
3.10 Experimental Setup . . . 81
3.10.1 System Configurations . . . 82

4 Experimentation and Evaluation **83**
4.1 Mutation Rate . . . 83
4.2 BGA . . . 85
4.2.1 Makespan . . . 85
4.2.2 ARUR . . . 92
4.3 SGA . . . 98
4.3.1 Makespan . . . 104
4.3.2 ARUR . . . 109

5 Conclusion and Future work **115**
5.1 Conclusion . . . 115
5.2 Future Work . . . 116

Bibliography **118**

List of Figures

1.1 Stakeholders in Cloud 2
1.2 Types of Schedulers in Cloud Computing 8

3.1 Schedulers in Cloud Environment 39
3.2 Single Point Crossover 56
3.3 Multi Point Crossover 56
3.4 Mutation Operation 60
3.5 Balancer Genetic Algorithm 62
3.6 Balancer Operation 63
3.7 Symbiotic Genetic Algorithm 69

4.1 Mutation Rate Analysis - A 84
4.2 Mutation Rate Analysis - B 84
4.3 Makespan on GoCJ Dataset 100 to 550 86
4.4 Makespan on GoCJ Dataset 600 to 1000 87
4.5 Average Makespan on GoCJ Dataset 87
4.6 Makespan on Left Skewed Dataset 88
4.7 Average Makespan on Left Skewed Dataset 88
4.8 Makespan on Right Skewed Dataset 89
4.9 Average Makespan on Right Skewed Dataset 89
4.10 Makespan on Normal Dataset 90
4.11 Average Makespan on Normal Dataset 90
4.12 Makespan on Uniform Dataset 91
4.13 Average Makespan on Uniform Dataset 91
4.14 ARUR on GoCJ Dataset 100 to 550 92
4.15 ARUR on GoCJ Dataset 600 to 1000 93
4.16 Average of ARUR on GoCJ Dataset 93
4.17 ARUR on Left Skewed Dataset 94
4.18 Average of ARUR on Left Skewed Dataset 94
4.19 ARUR on Right Skewed Dataset 95
4.20 Average of ARUR on Right Skewed Dataset 95
4.21 ARUR on Normal Dataset 96
4.22 Average of ARUR on Normal Dataset 96
4.23 ARUR on Uniform Dataset 97
4.24 Average of ARUR on Uniform Dataset 97
4.25 Makespan on GoCJ Dataset of 500 Jobs 105

4.26 Makespan on GoCJ Dataset of 1000 Jobs 105
4.27 Makespan on Left Skewed Dataset of 500 Jobs 106
4.28 Makespan on Left Skewed Dataset of 1000 Jobs 106
4.29 Makespan on Right Skewed Dataset of 500 Jobs 107
4.30 Makespan on Right Skewed Dataset of 1000 Jobs 107
4.31 Makespan on Normal Dataset of 500 Jobs 108
4.32 Makespan on Normal Dataset of 1000 Jobs 108
4.33 Makespan on Uniform Dataset of 500 Jobs 109
4.34 Makespan on Uniform Dataset of 1000 Jobs 109
4.35 ARUR on GoCJ Dataset of 500 Jobs 110
4.36 ARUR on GoCJ Dataset of 1000 Jobs 110
4.37 ARUR on Left Skewed Dataset of 500 Jobs 111
4.38 ARUR on Left Skewed Dataset of 1000 Jobs 111
4.39 ARUR on Right Skewed Dataset of 500 Jobs 112
4.40 ARUR on Right Skewed Dataset of 1000 Jobs 112
4.41 ARUR on Normal Dataset of 500 Jobs 113
4.42 ARUR on Normal Dataset of 1000 Jobs 113
4.43 ARUR on Uniform Dataset of 500 Jobs 114
4.44 ARUR on Uniform Dataset of 1000 Jobs 114

List of Tables

2.1 State-of-the-art Scheduling Techniques 30

3.1 Specifications of Tasks 36
3.2 Power Specification of VMs 36
3.3 Solution no.1 36
3.4 Solution no.2 36
3.5 Solution no.3 36
3.6 Calculations of Makespan and ARUR 37
3.7 Discrete Encoded Chromosome 42
3.8 Binary Encoded Chromosome 45
3.9 Fitness Values of Sample Solutions 53
3.10 Random Mutation Example 60
3.11 BGA Parameters 66
3.12 SGA Parameters 72
3.13 Presented Techniques and Performance Measures 74
3.14 Bechmark Techniques 78
3.15 Sample of Synthetic Dataset 80
3.16 Sample of Realistic Dataset 81
3.17 VM Specifications 81
3.18 System Configurations 82

4.1 Makespan and ARUR on GoCJ Dataset 100
4.2 Makespan and ARUR on Left Skewed Dataset 101
4.3 Makespan and ARUR on Right Skewed Dataset 102
4.4 Makespan and ARUR on Normal Dataset 103
4.5 Makespan and ARUR on Uniform Dataset 104

Abbreviations

ARUR	Average Resource Utilization Ratio
BGA	Balancer Genetic Algorithm
DSOS	Discrete Symbiotic Organism Search
ETA-GA	Efficient Task Allocation Genetic Algorithm
GA	Genetic Algorithm
MIPS	Million Instructions Per Second
MI	Million Instructions
MGGS	Modified Genetic Algorithm with Greedy Strategy
PSU	Percentage of Share Used
QoS	Quality of Service
RALBA	Resource-Aware Load Balancing Algorithm
SGA	Symbiotic Genetic Algorithm

Symbols

Avg_{PSU}	Cumulative percentage of share used
$Best_1$	Fittest chromosome among the whole population in GA
$Best_2$	Chromosome with second best fitness among the whole population in GA
CT_{VM_j}	Completion time of each VM
$load\_balancer\,(x)$	The fitness value computed against chromosomes of GA
$ShareRatio_{vm_j}$	Ratio to compute the share of each VM
$ShareUsed_j$	Share utilized by VM
$VMShare_j$	The share of each VM

Chapter 1

Introduction

This chapter provides an overview of the problem of task scheduling in area of cloud computing. The identified problem, applications, scope and contribution of research in cloud computing are discussed in this chapter.

In this modern era, the internet is widely used, and the applications of technologies have increased considerably all over the world. The internet based online applications are highly demanded by users [1]. With the expansion in the number of users, the data is exponentially growing [2], which requires the need of more computing resources. The immense use of internet resources and processing power require effective scheduling of user's tasks which is described later in this research.

The Cloud Computing is an emerging technology which ensures the delivery of cloud resources in terms of storage, processing, memory and applications to the end users [3]. Cloud computing is adopted by enterprises to perform computationally intensive tasks. The computing resources are distributed and accessed over the internet. There can be many stakeholders involved in cloud computing environment and some prominent stakeholders are cloud service provider, cloud broker, and consumer of the services [4]. Out of these, two major entities directly affected by task scheduling are cloud service provider and cloud service user. Both parties have their own interests whereas the optimal performance being a common advantage is essential for both sides. The cloud broker [5] acts as an intermediator

between cloud service provider and service consumer. The interaction between the cloud actors is shown in Figure 1.1. Cloud service provider strives to have good satisfaction level of their service consumer. Cloud networks provide services to user on demand; therefore, availability of resources is necessary [6]. Cloud service providers make sure that the cloud systems are fully functional and optimized enough to serve their valuable clients. There are three major classifications of cloud services which are provided in the form of hardware and software. These are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [1, 4, 7, 8], they are defined as follows.

1. In IaaS, the cloud infrastructure having networks, storage, processing and applications are provided to the user.

2. In PaaS, the platform for the development of applications is provided to the user via the internet. The developers can develop and test their applications on tools available in cloud machines.

3. In SaaS, the software applications are provided to the user remotely where the user does not have to have software in their own system for usage.

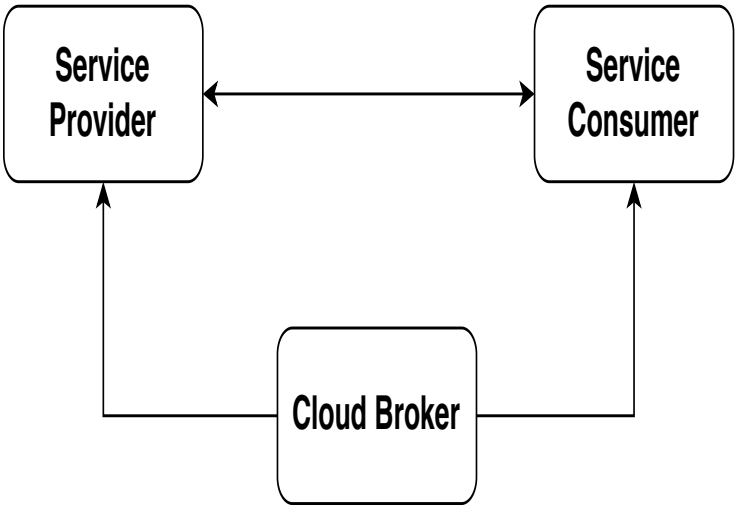


FIGURE 1.1: Stakeholders in Cloud

The on-demand provisioning of resources, resource pooling and multi-tenancy are some of the characteristics of cloud computing [9] which have attracted lot of users to the cloud computing. These characteristics are described briefly as follows:

1. Multi-tenancy: In multi-tenancy cloud service providers share a single physical resource. Same application can be shared among multiple customers.

2. Resource pooling: Same physical machine is used to serve lot of customers. This is done through multi-tenancy.

3. Resource provisioning: The resources are available for customers and more resources can be provisioned without having the intervention of service provider.

The cloud service providers strive for the best resource utilization which can ultimately improve the user experience by providing quality of services [2] complying with the QoS parameters. Lot of researches have been conducted in this regard to find the optimal algorithm for mapping of tasks to VMs. This research presents evolutionary approach intended to produce optimal mapping of tasks to VMs.

Research community is continuously working on scheduling algorithms in different domains. One of the areas where scheduling is vastly used is cloud based environment for the scheduling of cloudlets.

1.1 Heterogeneous Distributed Computing

The cloud resources are heterogeneous [5], which have different computational capabilities. These resources are in the datacenters which are distributed over the network [4, 8, 10]. Datacenters have physical host machines and the host machines are virtualized. The virtual resource in cloud is known as virtual machine (VM). The virtualization is vastly applied in cloud computing [7, 11] and it serves some major benefits in cloud computing environment. The virtual machine imitates the physical machine by providing insulated hardware and application [12]. Virtualization itself increases the resource utilization by avoiding the waste of actual

resource. Virtualization affords multiple VMs simultaneously and an application can be shared with multiple users. The cloud systems use the concept of virtualization to provide the resources and operations in much efficient way.

Heterogeneity is also at the tasks arriving in the cloud datacenters [7]. These tasks are usually of large size with varying requirements and constraints. The task size is defined in million instructions (mi). Jobs are mapped to cloud resources and this decision must be taken considering all the requirements of jobs and resources. Clouds have their own infrastructures and policies [13]. The cloud users get connected to cloud and they get services. The details of virtualized machines in cloud are hidden from the clients and the clients are more concerned about their tasks being executed efficiently. The cloud service providers want to do efficient utilization of the available resources. Clients and cloud service providers may have Service Level Agreements for provisioning of services [13] with any defined terms and conditions.

1.2 Scheduling Issues

Task scheduling refers to the mapping of jobs to the VMs. The problem of task scheduling in cloud environment is very vital because the cloud-based systems have to efficiently use the cloud resources, in a way that leads to minimum finishing time for the batch of jobs and maximum resource utilization. Task scheduler has to provide a good mapping of cloudlets to VMs [5, 7, 14–22]. The need for optimal mapping leads to meta-heuristic and heuristic approaches. There are many heuristic, meta-heuristic and hybrid solutions available in the form of scheduling algorithms [5, 7, 14, 23–30] but there is still need for improvement as the problem is NP-hard. The meta-heuristic approaches have more potential to explore the huge search space of possible solutions [28, 31] and they can be opted for getting the promising results. But heuristic approaches are only problem dependent and may not work well with different datasets [4, 30]. Therefore, optimization based meta-heuristics can be much useful to overcome the dataset dependent results of

scheduling [30]. Scheduling is done at both application level and Virtual Machine (VM) level. VMs must get the cluster of jobs which are best suitable for them, depending on some factors related to specification of VMs. Scheduling techniques must consider all significant parameters of VM which is the resource and cloudlets which execute on the resources. There is a need to have optimal mapping in which the load balancing is considered with the improvement in finishing time of jobs.

The load balancing [2, 18, 19, 24, 32–34] is the notion of distribution of workload among the heterogeneous resources in a way that neither a resource is over utilized nor underutilized. The overutilization is identified as some resources are used more than others during the execution of batch of jobs. The underutilization on the other hand means that some resources remain idle while other are busy in execution of batch of jobs [7, 13]. When there is underutilization then there exists overutilization too [2, 7, 14, 24, 35]. Having hundred percent full utilization that neither there exists overutilization nor underutilization is an ideal condition. But scheduling approach should reduce the overutilization and underutilization [21, 36–41]. Therefore, the balanced mapping of cloudlets to desired or most suitable virtual machine is challenging and ongoing area of research in cloud computing. Besides that, when the load is imbalanced there is another concept of virtual machine migration in which the overloaded machines are migrated to other physical server or machine [42, 43]. The load balancing is about balancing the load within the available resources without migration of VM to another machine. The VM migration is not under consideration in this research. There are different measures to evaluate the resource utilization and these measures are part of QoS parameters. ARUR is a measure that has been used extensively in the literature [7, 13, 35, 44–51].

The optimal scheduling is subjective as there are some parameters which are conflicting like if execution time is focused too much then the algorithm maps largest jobs to fastest VM and when response time is focused much then algorithm maps smallest jobs to fastest VM. Some parameters are conflicting, but a balance can still be created. Therefore, the optimal scheduling is subjective and depends on the selected objectives. The scheduling techniques can be single or multi objective.

In the literature many techniques are used for improving the optimization of task scheduling [2, 5, 8, 10, 14, 17–19, 23, 24, 32, 44, 52–60]. They need proper tuning of parameters to work for any problem. The meta-heuristics can be opted to optimization problem by appropriate problem encoding. Identification of optimal parameters for meta-heuristic helps to achieve the optimization goals. The objective function should be one covering the objectives of optimization like reducing makespan and improving load balance among virtual machines.

1.3 Taxonomy of Tasks

In cloud environment jobs or tasks are of varying nature and their processing requirements are different. Cloud jobs can be classified in categories of dependent and independent jobs, according to their nature. The independent jobs can be executed without depending on other jobs. The dependent jobs are normally referred as workflow and they have priorities. It can further be classified as preemptive and non-preemptive. The preemptive jobs can be interrupted during the execution and their execution may be resumed later. Jobs may have deadline constraints and other Service Level Agreements (SLA) [13]. The SLA are negotiated between the customer and service provider which enforce to comply with some stipulated Quality of Service (QoS) parameters. They need separate heuristic and meta-heuristic problem descriptions and constraints. In priority-based mapping the scheduling decision is biased towards the priority of jobs and jobs are usually preemptive. This research provides mechanism of mapping of jobs to VMs where jobs have no priority or deadline [45, 61]. Having no deadline does not really mean that timely execution of job is not important. This research is intended to work on independent, non-preemptive at application level and static natured task scheduling in cloud computing environment.

The Quality of Service (QoS) parameters [62] for cloud providers and cloud users must be assured in a good cloud-based infrastructure. There are different QoS parameters used for the evaluation of services provided to the consumer of services.

Some QoS parameters are makespan, response time, cost, resource utilization, throughput and energy consumption. These and other parameters must be focused to improve user and service provider experience.

1.4 Scheduling Techniques

Scheduling techniques can be static or dynamic [63]. In static scheduling techniques the information about the jobs and resources is available prior to the decision of scheduling [64]. Therefore, the decision is taken at the compile time. On the other hand, in dynamic scheduling there is no information available about jobs and resources prior to the execution of jobs. The dynamic scheduling can also be called as online nature scheduling where the jobs arrive, and the scheduling is done considering the previous states and estimations. In static scheduling the jobs are received in batch before the scheduler decides their mapping.

Task scheduling techniques used in cloud computing environment can be categorized as heuristic, meta-heuristic and hybrid approaches. Heuristic approaches are made to achieve certain objectives and they comply merely with the specified criteria. Based on some criteria the heuristic jumps to a solution. They do not have the potential to explore further solutions unlike the meta-heuristics [4]. Heuristics find fix solution with no further chances of improvement. Heuristics are problem dependent and they cannot be applied to other range of problems. Whereas the metaheuristic can be applied to vast range of optimization problems, particularly when there are many possible solutions [30]. Meta-heuristics do need problem dependent tuning and fitness criteria. The metaheuristics have the potential to explore and exploit large search space to find better solution than the solution discovered so far. The metaheuristics promise to find solution close to optimal solution [34]. They are choice of researchers from years to solve optimization-based NP-hard problems. The hybrid approaches have combination of heuristics, metaheuristics or both [34]. The meta-heuristic and heuristic can work good in

combination to overcome the vulnerabilities in the search process. They can be combined in the following possible ways:

- Heuristic combined with other heuristic(s) [16].
- Metaheuristic merged with other metaheuristic(s) [10, 30, 39, 65, 66].
- Metaheuristic with heuristic(s) in main algorithm [14].
- Metaheuristic having heuristic(s) in initialization phase only [67].

Figure 1.2 shows some prominent type of schedulers used in cloud computing environment. The static scheduler having hybrid techniques are focused in this research.

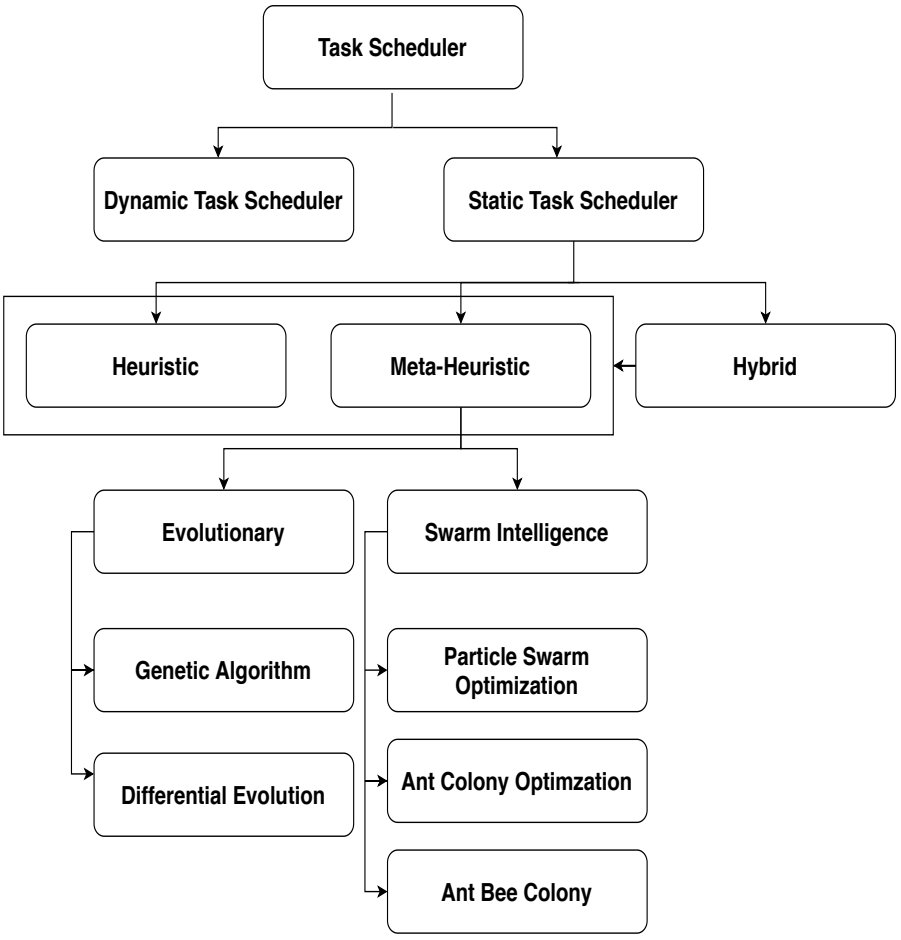


FIGURE 1.2: Types of Schedulers in Cloud Computing

1.5 Evolutionary Algorithms

The evolutionary algorithms [68] are techniques for finding the optimal solution. In evolutionary approaches the algorithm strives to find better solution in each iteration and the definition of better solution depends on the defined objective function. The generic steps involved in evolutionary techniques are: Initialization, evaluating the fitness of solutions, selection of best solution, updating the population and termination. When the algorithm reaches to a stage where there is no more improvement in the fitness value achieved so far then the algorithm is considered to be converged. The convergence is no guarantee that the best solution has been discovered. The evolutionary approaches may take lot of time to reach to the best possible solution and have their own capabilities to explore and exploit the search space [2, 4, 8, 28, 30, 34, 52, 53, 69].

1.6 CloudSim

CloudSim [70] is a simulation framework and a tool developed for cloud computing environment. It was built in CLOUDS laboratory in the University of Melbourne, Australia. It is the most popular choice of testing the cloud-based scheduling algorithms. The advantage of using cloudsim is that the user of CloudSim can focus on the specific problem without being worried about the details of cloud-based infrastructure as it is being handled in the classes of cloudsim. This provides an environment of real cloud system where user can test the performance of their proposed model. The scheduling algorithms can be used at different levels in CloudSim library:

1. Scheduling can modify the mapping of VMs to datacenters and this is handled in DatacenterBroker class.

2. Scheduling can decide the host machine for placing new VM. This is provided in VmAllocationPolicy.

3. VmScheduler is responsible for deciding the allocation of VM within a single host machine.

4. CloudletScheduler receives the mapping of cloudlets to VMs and scheduling algorithm is responsible to define an optimal mapping.

This research is related to the scheduling of tasks to VMs which is being handled in CloudletScheduler. While the other mechanisms like scheduling of jobs within a single VM and scheduling of VM to physical machine have their own algorithms in cloudsim. They can be set to default and task scheduling is only related to scheduling of heterogeneous tasks to heterogeneous VMs.

CloudSim has layered structure having Network, Cloud Resources, Cloud Services and User Interface layers. These layers include the three models of services which are IaaS, PaaS, and SaaS.

1.7 Problem Statement

In cloud computing the task scheduler needs to find an optimal mapping of jobs to VMs, considering the makespan and load balancing, from a huge search space of possible mappings. A few of existing techniques have tested their approaches on large datasets based on real traces of cloud. For better task scheduling, the existing schedulers using different meta-heuristics can be further improved and it is required to achieve an early convergence in addition of better load balancing.

1.8 Research Questions

Following are the research questions derived from the literature, considering the research gap:

Q1: How load balancing and makespan can be improved for efficient task scheduling in cloud computing?

Q2: How faster convergence can be achieved in the area of task scheduling?

Q3: What is the effect of fusion of heuristic and meta-heuristic for task scheduling in cloud computing?

1.9 Purpose

The existing heuristic and meta-heuristic techniques have given solutions to this problem for better task scheduling and there is further space of improvement [4, 14, 27–29, 65, 71–73]. The problem is that the cloud jobs are becoming more complex and the size of jobs is increasing in terms of millions of instructions. The focus of this research is to find an optimal mapping of cloud tasks to VMs, while considering that the hardware resources are limited, and they must not be wasted even for few seconds.

The genetic and symbiotic organism search algorithms have potential to achieve the good task scheduling [5, 14] as an optimization problem. Their advantages can be combined to overcome the vulnerabilities of these techniques and other approaches. This research provides mechanism of combining the benefits of these two algorithms with proper structuring and tuning of parameters. The meta-heuristic adopted to a problem has to be properly tuned for making good balance between exploration and exploitation. This research briefs the optimal settings to balance exploration and exploitation for getting good results of task scheduling optimization.

1.10 Scope

This research is intended to facilitate the efficient use of cloud resources which would be helpful to cloud service providers and the cloud users to better meet their requirements. This research work provides necessary guideline to improve task scheduling using hybrid meta-heuristic approach. The parameter tuning and

hybridization study of metaheuristic provided in this research can also be adopted to other problem domains. The benefits of meta-heuristic through comparison of performance metrics listed in QoS are demonstrated in this research. QoS parameter for clients focused in this study is makespan and for cloud providers are both resource utilization and makespan. Improving QoS of task scheduling with optimization can lead to better results by exploiting and exploring the search space of possible solutions in cloud computing.

1.11 Significance of Solution

This research work contributes in efficiently reducing the makespan and balancing the workload among heterogeneous cloud resources in cloud computing. The convergence of genetic based scheduler is enhanced which can be embraced in other applications of genetic algorithm.

1.12 Dissertation Organization

The remaining sections of the dissertation are organized in the following pattern. Chapter no.2 describes the review of the relevant literature. The procedures adopted by other researches and the shortcomings are highlighted which led to the presented solution. In chapter no.3 the steps to reach to the presented solutions are portrayed. The methodology and architecture of the presented solution alongside the dataset are exhaustively described. Chapter no.4 demonstrates the detailed experimentation conducted for evaluating the presented work. Whereas the chapter no. 5 provides the conclusion and directions for future study related to this research.

Chapter 2

Literature Review

This chapter provides the literature study of related work and highlights some key issues that led to the proposed solution. Lot of techniques have been employed for independent task scheduling in the domain of cloud computing [2, 4, 5, 7, 8, 10, 11, 14–30, 32–34, 36–41, 44, 52–60, 65–67, 69, 71–88]. The literature review chapter thoroughly analyzes all possible related techniques which can be heuristic, metaheuristic or hybrid.

2.1 Heuristics

He et al. in [74], introduced an improved min-min algorithm for task scheduling in cloud, based on QoS like completion time, reliability and cost. It requires cloud resources to provide its QoS priorities. Scheduling is divided in high and low QoS queues. The low QoS queue is scheduled after the high QoS queue. The min-min algorithm [15] improves the makespan for small sized jobs. The idea is to assign smallest job to fastest VM in order to reduce the maximum completion time, in this way the task with minimum execution time is mapped to a fastest resource [15, 16, 24, 41, 74]. The load balancing is not under consideration during scheduling of tasks. Therefore, the load is imbalanced, especially in case when the jobs are of very large sizes. The LBMM in [16] uses the traditional min-min algorithm in

first phase then the jobs of overloaded VM are remapped to underloaded VM in second phase. The overloaded VM is chosen based on makespan. The task with minimum execution time in overloaded VM is selected. The maximum execution time of that task is calculated against each VM and if it is less on any VM the task is shifted to that VM.

In [80] a max-min task scheduler was proposed to improve the resource utilization and response time. The max-min algorithm assigns the largest task to VM where task has minimum execution time to improve makespan but max-min performs well only when large size tasks are much in number. In [84] an improved max-min was proposed. According to author the execution and waiting time are further reduced through changing the selection criteria of completion time with the execution time. An enhanced max- min was proposed in [83]. In enhanced version the selection criteria differs from the traditional max-min in a way that the largest job is mapped to VM which has nearest to minimum completion time. The enhanced version is evaluated on a very small set of jobs.

An attempt to improve the min-min for resource utilization and makespan was made in [41]. After the execution of min-min the balancer operations are triggered in the proposed technique. In the balancing heuristic the tasks are swapped in resources where they produce minimum execution time. Author in [41] tried to achieve the load balancing using the minimization of makespan which can be further improved. The heuristic for balancing takes too much time in worst case because search space is huge, and the proposed technique continues unless the load balancing condition is met. In [88] the performance of min-min and max-min was analyzed, and it was found that max-min performs better when large size tasks are much in number whereas for the opposite scenario min-min outperforms [80, 88].

ETSA [22] a heuristic was presented to improve the energy consumption and makespan. It considers the computation time of tasks and utilization for mapping. The energy consumption is targeted by improving the resource utilization. The ETSA consists of three phases. The first phase is the estimation phase in which the computation time based on expected time to complete and resource

busy time are calculated. In second phase which is of normalization and selection of minimum value of objective. The third phase assigns the task to VM producing minimum value just like balancing in [7, 41]. As the utilization is computed merely based on the time the resource was busy. Therefore, this criteria of improving resource utilization does not specify the necessary details which can contribute in optimal resource utilization.

In [77] two statistical based heuristics were proposed called as HMADF and Suffrage-HMADF. HMADF computes the mean absolute deviation of each tasks. Then the task having highest mean deviation is assigned to VM having earliest completion time. This is how small and large jobs are mapped earlier, to improve the execution time. The suffrage-HMADF considers the QoS information of tasks and rank them as high or low QoS tasks. The high QoS are mapped with the suffrage and the low QoS are mapped with HMADF. The large jobs are those which deviates from the mean, but this is problematic when the dataset is left or right skewed. It generates imbalance of resource utilization. The suffrage algorithm based on execution time (ETSA) was proposed in [37] to improve the execution time as well as the resource utilization. The execution and completion times are computed. The suffrage value for both completion and execution time is calculated by subtracting first minimum from the second. The task is assigned to VM based on the minimum suffrage value between execution and completion time. The ETSA does not have mechanism to assist the load balancing. Although execution time can improve the load balancing but not to a significant extent.

In [36] cloudlet scheduler based on travelling salesman approach (TSACS) was proposed. TSACS converts the task scheduler to TSP domain before the application of TSP based solution. TSACS has three different phases. The first phase is the clustering phase in which the large size problem is converted to small sized clusters to reduce the computational complexity of the problem. The number of clusters is equal to the number of VMs. Then in the second phase problem is converted to TSP. In third phase known as the assignment phase, the nearest neighbor approach is deployed to assign jobs to VMs. The nearest neighbor technique begins with the random city and keep visiting nearest cities till all are visited. The TSACS

is supposed to improve the load balancing but there is no direct information of resources or steps involved in balancing. Therefore, TSACS emits imbalanced load among the resources.

Panda in [33], proposed a probabilistic load balancing (PLB) scheduling algorithm to schedule the tasks while considering the load on available VMs. The PLB is a two-phase technique. It begins with the sorting of task in non-decreasing order. The initial load of each VM is calculated. In the second phase the minimum initial load is chosen to map the successive task to that VM followed by the updating the initial load. Even if the load is balanced the execution time may still improve for the same value of load balancing but the PLB does not consider the execution time of tasks. Whereas, PLB improves the load balancing. The PLB approach benefits the underloaded VM by mapping more tasks to it. But there is nothing done for the overloaded VM to reduce the load.

RALBA [7] calculates the share of virtual machines based on the total size of jobs and available computational power of the virtual machines. It has two schedulers namely, fill scheduler and spill scheduler. The fill scheduler selects the largest VM on the basis of largest VM share and makes a pool of possible jobs that can be mapped to that VM. The job sizes should be less than or equal to the largest VM share. Then the largest job is mapped to the VM with largest VM share and the VM share of that VM is updated. The next largest VM share is selected and the spill scheduler continues until there is no smallest cloudlet less than the smallest VM share. The remaining cloudlets are mapped through the spill scheduler. Spill scheduler assigns jobs to VMs based on Earliest Finishing Time calculated against each VM. RALBA has proven to be better in load balancing and makespan as compared to RASA, TASA, and other heuristic techniques [7, 35]. But results of RALBA are not compared with any meta-heuristic approach. Since the cloudlets are heterogeneous and there could be many possible assignments based on VM share. RALBA is not doing optimization and other possible assignments on VM share are not checked in RALBA. Therefore, there is still room of improvement in load balancing and also the load balancing is not so good when batch size is

small and standard deviation among job sizes is less. The load balancing does not always guarantee good makespan and makespan can be further reduced.

2.2 Meta-Heuristics

In [73] an improved PSO (IPSO) was proposed. The proposed technique deploys simulated annealing to enhance the convergence of PSO. To further improve the optimization and convergence the SA updates the particles, and this happens in every iteration. The two optimization approaches work in combination and PSO based particles are updated so there are chances that particles will loss the properties of PSO after the modification. The multi-objective PSO (MOPSO) was proposed in [82] to minimize the execution time and cost. The SPV rule is used in MOPSO to convert discrete values from continuous in PSO. An archive is maintained to list the dominated and non-dominated particles. The performance of MOPSO is evaluated with a very small set of tasks. Fahimeh in [60] proposed a multi-objective PSO to improve time and cost of execution. The same objectives were considered in [82]. An archive of particles is maintained where the dominating ones are deleted, and non-dominating are added. Then they are sorted based on the objectives they are improving. Out of those sorted, best particle based on fitness are chosen to be a global best. The load balancing is not under consideration. Hybrid PSO algorithm was proposed in [57] having Differential Evolution algorithm. DE is used to update the velocity in standard PSO. This helps to avoid premature convergence in PSO. The particle position is only updated when it produces better fitness. The fitness is defined on the base of makespan and resource utilization. The performance is evaluated on a very small dataset.

The [67] suggests that the merger of SJFP heuristic in PSO improves the performance of PSO. The SJFP initializes the population of PSO. Unlike [57, 73] a heuristic is used in [67] but only for the initialization phase. The proposed technique outperforms the standard GA and PSO for makespan as fitness function. Whereas no load balancing mechanism is deployed in it. Ageena in [59]

proposed an Integer PSO with the objective to minimize cost and makespan. The same objectives were considered in [60, 82]. The standard PSO is transformed to produce integer results by using modulus operation. The load balancing is not targeted in this work which results in imbalance distribution of workload among the VMs. The improved PSO was presented in [79] to enhance the utilization of resources. According to the author the traditional PSO suffers slow convergence in task scheduling. The simulated annealing was combined with PSO to take advantage of strengths of both techniques. The traditional PSO works with personal best but in this improved version the personal worst was introduced as a parameter to reach to a better solution quickly. The global best is found with the SA approach in the main PSO algorithm. SA was also deployed in [73] to improve the convergence and the proposed technique speeds up the convergence. It has been generally observed that speeding up the convergence often creates imbalance in the local and global search. The results are not compared on the basis of makespan or resource utilization as claimed and also the fitness function is not defined as multi-objective approach.

Another PSO based scheduler was proposed in [78], where the fitness is composed of power, storage and processing time. According to the author the time taken by the largest cloudlet is reduced to balance the load. However it does not consider actual load and hence there are chances of improvement. Greedy PSO was proposed in [39], with the intention to have better convergence, global search and balance among resources. The slow convergence of PSO is also reported in [66, 73, 79] which suggested the merger of meta-heuristic to improve it. The greedy approach deploys the load balancing by mapping task to VM with less workload. The population in PSO is initialized with the greedy technique. The proposed technique may converge too much fast due to guidance of initial population with heuristic, which can lead to premature convergence. Later MDAPSO was proposed in [66] which is an enhancement of DAPSO by merging Cuckoo Search algorithm. The reason for the merger of two algorithms is to improve the convergence and global search capabilities. According to the author when inertia weight reduces the exploration is compromised therefore CS has been merged. CS works after the final

iteration of DAPSO. The two optimization techniques are fully functional, making the technique computationally expensive. CPSO in [65] was proposed, having the combination of particle swarm optimization PSO and cuckoo search CS technique. CS was also deployed in [66]. The objectives considered in CPSO are makespan, budget cost and deadline violation. The CS is inspired by the behavior of cuckoos laying egg in the randomly chosen nest of host. The host dispose of the egg on finding them or leaves the nest. Here egg is the representation of a solution. To find the location the levy's flight known as random walk is chosen. Both PSO and CS are fully functional to find new solutions then the two newly formed solutions are transformed to one hybrid solution. The CPSO does not consider the load balancing as an objective of optimization.

In [40], LBMPSO the load balancing mutation algorithm based on PSO was proposed. The algorithm addresses the issues of VM where task are unallocated, multi-allocation of same task and slow convergence in PSO. The performance of LBMPSO was evaluated on a very small dataset. In [30] a PSO based task scheduler was proposed. The proposed technique deploys the load balancing mechanism in the PSO to improve the resource utilization and makepan. The load balancing is achieved using the honeybee model. The overloading is paralleled with getting honey from empty source. The standard deviation represents the load balancing of VM. The resource usage pattern is observed to move the tasks in overloaded VM to underloaded. The resource utilization mechanism in the proposed technique uses the information of processing time which does not consider the actual power. Therefore, the load balancing is not significantly good.

Zhao in [58] proposed PSO with inertia weight to improve the makespan. The fitness is defined on the basis of cost and time, whereas, load balancing is not considered. The adaptive weight is introduced to balance the global and local search of the particles. Thanaa in [10] introduced Binary LB-PSOGSA for load balancing, which is the combination of two algorithms, PSO and GSA. In PSO the particles communicate with each other to reach to global optimum and in each iteration the particles update their local best and overall global best positions and GSA has masses treated as search agents. The greater masses are considered better than

the others. The velocity of particles is updated using the combination of PSO and GSA. This combines the exploration capabilities of GSA in the exploitation capabilities of PSO. According to [66] there is need to improve the search capabilities of standard PSO and therefore a mechanism of merging techniques was proposed just like [10, 73, 79]. The binary encoding is used for representation of particles of PSO. The execution time is calculated as an objective function. Load balancing is not the objective function but PSOGSA claims to improve the balancing of load among VMs.

A PSO with inertia weight strategy was proposed in [76]. An attempt to improve inertia weight was also made in [58] to improve the makespan. To enhance the global search ability of PSO dynamic inertia weight was introduced by adopting the logarithmic decreasing approach. The mentioned technique improves the makespan when compared with ABC, DA and GSA. The inertia weight reduces over the iteration to assist the exploration of much search space. The logarithmic inertia strategy is the key factor in adding the exploration capabilities in the PSO. The logarithmic inertia strategy is compared with other four inertia weight strategies and comparatively logarithmic strategy outperforms in terms of convergence speed. The performance is evaluated on a very small dataset which may not be the right choice to exhibit the actual working of the technique. There is no consideration of load balancing, resulting in bad utilization of resources. An Integer-PSO was proposed in [25], with the objective to improve the load balancing and convergence of PSO. To retain the properties of standard PSO producing continuous values a mechanism to convert continuous values to integer for PSO is introduced. Same type of mechanism to convert SOS to DSOS was presented in [5]. Later a function is invoked to eradicate duplicate assignment of tasks. There is no mechanism supporting the load balancing in the objective function. Therefore, it is inferred that the Integer-PSO generates imbalance among the resources as load balancing is not processed considering the power of VM. In [34], Najme proposed a hybrid FMPSO technique to improve the throughput and load balancing. The FMPSO technique uses the fuzzy theory and modified PSO algorithm.

Four different velocity updating methods are opted in FMPSO. The initial population is generated using SJFP technique. The roulette wheel selection operation of evolutionary algorithm is used for introducing diversity in the selection of VM in SJFP. Then the global search capabilities of crossover and mutation operations of evolutionary algorithm are added to escape from trapping in local optima. The particles generated using both crossover and mutation are passed to PSO. The idea of this diversification mechanism is to overcome the shortcomings of PSO. Fuzzy theory is used to evaluate the fitness of particles. Due to inculcation of SJFP in initialization phase the solutions are more biased toward the particles supporting the fast execution of shortest jobs. The convergence of FMPSO improves due to global search abilities.

A bee life algorithm (BLA) was proposed in [85] to efficiently reduce the makespan when compared with GA. BLA is a nature inspired algorithm having the behaviors of food search and reproduction. In reproduction the crossover and mutation are applied like one applied in [52]. After the reproduction operation the bees search the neighbor regions for food. A greedy approach is introduced for the local search in BLA. The performance is evaluated on a very small dataset. In [81] an improved differential evolution-based task scheduler IDEA was proposed. Author in [81] proposed that hunting behavior of honeybee can balance the load in task scheduling. Here tasks are removed from the overloaded VM and then assigned to suitable underloaded VM. The cost model is composed to define the fitness. The Taguchi model is utilized in IDEA. According to the author the introduction of mask mutation has contributed in generation of better offspring. The performance of IDEA is evaluated with two test cases, but no large dataset is employed.

Shaminder in [17] presented an efficient GA. It uses LCFP, SCFP and randomization for population initialization. The fitness of chromosome is evaluated based on execution time. The crossover operation used is multi-point and mutation is based on swapping of genes. The algorithm improves makespan, but it is evaluated on a small dataset. LAGA [24] uses min-min and max-min in GA to improve the load balancing. The initial population is generated and has one chromosome created using min-min and one with max-min to take advantage of both minimum

execution time and avoiding late execution of large jobs. Rest of the population is generated using the randomization function. Proportionate selection is used to choose the parents. The single point crossover is used in LAGA. Total time of resources is considered in the time load balance model. The fitness is composed of both makespan and load balance value. The convergence of LAGA improves due to direction provided by chromosomes created using min-min and max-min algorithm. Shekhar in [56] presented a modified GA for scheduling independent tasks. The initial population is generated using enhanced max-min, where largest task is mapped to VM with smallest computational power, but the selection criteria for enhanced max-min is average execution time. The modified GA works better in terms of makespan with initial population generated through enhanced max-min. Tingting in [53], introduced a load balancing based genetic algorithm named as (JLGA). The initial population is generated following a greedy approach to reduce cost. Fitness is calculated based on total time including disk I/O and remote data transmission time. The load variance formulation is used to get better load balancing. Total time and load variance collectively define the fitness of any chromosome. Fitness ratio is used to guide the selection process. Furthermore, adaptive probability of crossover and mutation have been introduced to enhance diversity in population.

Javanmardi in [55] proposed the Hybrid GA having fuzzy theory. Jobs are assigned to resources based on bandwidth and computational power of resources. Fuzzy theory calculates the fitness, which is also used in crossover operation. The priority of parameters is defined on the base of fuzzy rules. Two chromosomes are selected through fuzzy theory and they are used to participate in reproduction of population. Instead of using the traditional single or multi-point crossover HGA uses the job length and VM mips as input to the fuzzy system. The genes of chromosomes are exchanged based on fuzzy output. The performance of HGA is dependent on the strength of fuzzy rules which may not work with all types of datasets. The convergence of HGA improves due to inculcation of fuzzy system. Zhenzhen et al. in [32] proposed a multi-objective GA. The objective function is designed to have effect of load balancing and completion time of tasks. The share

of load is not considered while manipulating the balance of load. The static rate of mutation and crossover are selected to achieve better diversity in consequent generations of population. Coherent Genetic algorithm (CGA) was presented in [44] to improve the execution cost and time. The cost is based on execution time and the amount of data transfer. The fitness value based on the above-mentioned cost formulation helps to achieve better load balancing. However, load balancing is not directly involved in the objective function. The performance is evaluated based on makespan and resource utilization and CGA improves than one of the improved versions of GA. Safwat in [19] proposed a TS-GA based scheduler in which fitness is evaluated based on completion time. The tournament selection is used in which the solutions not selected reserve their seat in the next generation. The performance of TS-GA is compared with the round robin heuristic which shows better resource utilization and makespan.

Zarina in [18] presented a RMGO for load balanced scheduling. The RMGO is combination of three algorithms namely min-min, max-min, and suffrage. These three algorithms run initially for generating the population. The best particles of these three algorithms become the input of the genetic algorithm. The makespan is considered as the objective function of GA and the roulette wheel is used as selection operation. The algorithm improves makespan and load balancing is not directly considered in the objective function and while updating the chromosomes in next generation. Kairong [8] presented an AIGA task scheduler. AIGA introduces adaptive probability of crossover and mutation operations. AIGA does not consider the load balancing. The position of chromosomes in the population, number of iterations, population size, fitness of best chromosome and a threshold value are used to compute the mutation and crossover rate in AIGA. The AIGA improves in performance as compared to the standard GA. Improved Genetic Algorithm (IGA) was introduced in [23], to maximize the resource utilization. The resource utilization is focused by monitoring the resources which are idle during task execution. Initially the jobs are assigned to idle resources until there are no more idle resources. The remaining jobs are assigned to resources whose expected

completion time is less. Cost is calculated and used as fitness function. The resource utilization for the chromosomes are checked and chromosomes are updated based on that. The chromosomes are updated based on the resource utilization. The performance is not evaluated on any large-scale dataset.

In [14], Zhou et al. have proposed modified genetic algorithm (MGGS), which considers the total completion time of jobs and load balancing. The MGGS takes less iterations to converge due to the greedy strategy guiding the optimization process by updating the chromosomes. The iterations of MGGS take too much time when the batch size increases. The MGGS uses binary encoding to represent the candidate solutions. The fitness function is made up of maximum completion time to achieve minimum completion time for the batch of jobs. Roulette wheel is used for selection of best particles. The probability of crossover and mutation is defined using equations. The greedy strategy is used to update all the population in each iteration. In greedy strategy the total execution time of each VM is calculated. VMs are sorted according to the execution time and VM with maximum execution time is selected. The minimum job in that VM list is eliminated and added to VM with minimum execution time. VM list is updated again according to the execution time. This greedy approach continues until VM with minimum execution time becomes the VM with maximum execution time. In this way the imbalance is reduced by greedy strategy. The MGGS algorithm achieves good makespan and load balancing. The algorithm converges very quickly but get trapped in local minima and it can be further improved. In worst cases the MGGS takes so much time to balance the load and balancing operation continues until the VM with minimum execution time becomes the VM with maximum execution time. Too much guidance to the GA algorithm makes it converge faster but the chromosome losses their actual properties to better find the optimal solution in the large search space. The MGGS algorithm is tested on a very small set of jobs and it should be tested on large-scale dataset to better get the idea how algorithm is performing with real cloud traces.

In [52] Rekha introduced an ETA-GA based efficient scheduling algorithm. The algorithm computes the fitness of chromosomes based on total completion time

and probability of failure. The failure is the network delay occurring among the network nodes and the fitness is the minimization function. The ETA-GA selects two chromosomes through the selection of best among the population based on the fitness value. It means the best chromosome has the maximum probability of selection to get involved in reproduction operation. Crossover is selected as multi-point with the constraint of probability. The selection of best in each generation leads to over exploitation and the algorithm suffers pre-mature convergence. Here better resource utilization is achieved through minimizing the finishing time. But the resource information of load balancing is not part of objective function which leads to poor resource utilization. The Hybrid GA with Ant Colony algorithm (HGA-ACO) was proposed in [54], which is the combination of two different meta-heuristics. The utility scheduler is used to order the tasks in queue based on memory and execution time. The output of utility scheduler becomes the input of HGA-ACO. The best chromosomes are passed to Ant Colony algorithm where they are transformed using the path. Path is evaluated through response time and completion time in ACO. The resultant chromosomes get involved in crossover and mutation phase of GA. The load balancing is not considered as an objective and the overhead of two meta-heuristic slows down the algorithm. The best particle obtained through GA is used in crossover and mutation after transformation occurred in ACO phase. Fang in [2], proposed an improved GA encoded in binary format to better use the good search ability of binary representation. The load balancing of chromosomes is also calculated. The load balancing is computed through formulation of standard deviation (SD). The SD represents deviation of the expected time to complete the tasks on computing resources. The fitness value is composed of total time and load of tasks. The probability of selection of fittest chromosome for generating offspring is based on proportionate selection to maintain diversity in the population. The probability of crossover and mutation is adaptive which changes automatically when fitness among chromosomes is scattered or constant. The improved GA works better when tasks are much in number. The vector size grows significantly with the increase in number of jobs. The gene representation in binary encoding becomes more complex with the increase in jobs. In [38] the

Imperialist Competitive Algorithm (ICA) was deployed to improve the execution time of tasks. According to author the ICA has slow convergence speed. Therefore, to improve the convergence and search of ICA the GA is applied in updating the new positions. The performance is evaluated on a very small dataset which is not based on real traces of cloud. Therefore, the technique may have different behavior on large dataset.

DSOS was proposed [5] a discrete version of SOS to optimize the task scheduling problem for independent tasks. The initial population consisting of organisms are generated randomly. Vectors consist of 1xn dimension where each value is the VM number corresponding to job and 'n' represents the number of jobs. Best organism is selected based on the makespan. Organisms are updated using mutualism, commensalism and parasitism operations. Mutualism mimics the behavior of symbiosis relation between two organisms where both get mutual benefit from each other. Commensalism is a relation between organisms where one gets the benefit and other is neither benefited nor harmed. Parasitism is a relation in which one organism gets the benefit and other gets harmed. Population consisting of organisms is updated using the above-mentioned symbiotic operations until the stopping criteria is met. The best organism with fitness as makespan is selected as the final mapping scheme. SASOS [38] simulated annealing based SOS was proposed to enhance the SOS algorithm. According to author the acceptance of best solution leads to fast convergence, but here it is observed that due to selection of best chances of trapping in local minima increases. The SA is used to enhance the local search of SOS and to speed up the convergence. The SASOS stimulates the improvement in makespan and load balancing. The SASOS does not directly incorporate any balancing mechanism, hence not contributing to resource utilization. In [20] an improved hybrid SOS was presented, where the actual SOS algorithm is lessened, and two more algorithms are combined. The CLS algorithm is used to improve the convergence and the SA to enhance the exploration. The objective function is based on resource utilization calculated through maximum and minimum utilization of resources in terms of makespan, whereas the actual load is not

considered. The SA is used to update the organisms in mutualism and commensalism phases of SOS. The best organism is updated using the CLS algorithm. In CMSOS [4], the symbiotic organism search algorithm is used with the combination of chaotic optimization strategy at the initialization phase of SOS. The random sequence generation in SOS algorithm is replaced with the chaotic strategy to ensure more diversity in the population with an idea to improve the global search capability of SOS algorithm. The CMSOS improved the makespan and financial cost of the task scheduling. The initial population is generated using chaotic local search sequences and in each phase of SOS the chaotic sequences are generated. The ecosystem for next generation is selected through non-dominated sorting and selection by crowding distance which is the average of the neighboring solutions. As the chaotic search sequences are used in each generation therefore, the organisms created using SOS have high probability to be replaced by the chaotic sequences. Therefore, the organisms having properties of symbiotic relations are more likely to lose in the proceeding generations of the algorithm. It is not always beneficial to maintain the diversity. When non-dominating solutions are more than the size of the ecosystem then the crowding distance is applied to select solutions equal to size of ecosystem. This causes loss of some local best organisms which may contribute in the search of global best. The imbalance of workload has no role in defining the fitness of the solution, resulting in the imbalance distribution of tasks to VMs.

In [72] the cat swarm optimization algorithm (OTB-CSO) based on Taguchi was proposed to improve the execution time. The Taguchi optimization uses the orthogonal matrix representation. The Taguchi approach is incorporated in CSO to improve the local search capabilities. In CSO the population comprising of cats is generated randomly. The fitness is evaluated and then the seeking and tracing modes are applied accordingly. The seeking mode is for ensuring the global search having the mutation operation, whereas the tracing mode is for local search. Both modes in CSO updates the position and velocity of cats. The performance of the proposed technique is evaluated on a very small dataset and the load balancing is not considered. The Lion Optimization Algorithm (LOA) was proposed in [29] for

task scheduling. The LOA is inspired by the behavior of lions exhibiting the mating, hunting and defense mannerism of lions. The population in LOA is divided in prides and each pride has some percentage of male and female lions. In this technique the fitness is based on makespan. The best position achieved in all these behavioral phases is considered the best solution over the iterations. As the load balancing is not an objective so therefore the load is imbalance. The performance of LOA is not evaluated on any realistic dataset.

The WOA scheduler was proposed in [28], and it is based on whale optimization technique. The WOA was designed to comply the objective of reducing makespan and cost. The algorithm begins with the assumption that the initial solution is the best one. The fitness of solutions is computed then the position of surrounding prey is considered. The search agents revise their positions based on the knowledge of best position acquired. The WOA technique does not employ any load balancing mechanism which makes the optimization unenlightened in creating the balance among resources. In [27] HGDCS was proposed based on gradient descent cuckoo search. According to the author the cuckoo search (CS) has entities which have identical search conduct, leading to the discovery of local optima instead of global optima. Thus, gradient descent (GD) approach is merged with CS to help escape the local minima in search of optimal solution. The GD helps find the local minima of a function by moving in the proportional direction of the positive gradient. The gradient method is applied in the CS optimization to improve the local search abilities. The HGDCS converges very fast but it does not balance the load among resources. The local search is greedy which generates imbalance between the local and global search, resulting in deprived optimization. The CSSA was proposed in [21]. It is inspired by the chaotic social spider and the foraging process. It reduces the makespan and improves load balancing. The algorithm begins with the random population. The search agents (SA) broadcast messages and the best of which is chosen. The CSSA adopts the chaotic inertia weight in VM selection process. The load factor parameter is introduced to avoid the selection of overloaded VM. Finally, the new position of the SA is updated. The CSSA

supports the assignment of tasks to underloaded VM but it does not consider the actual computational requirements of VM in process of load balancing.

In [69] a hybrid task scheduler (MSDE) was proposed. The MSDE is based on improved moth search algorithm having the differential evolution algorithm. The moth algorithm is inspired by the phototaxis and the levy flight in the nature. The phototaxis exhibits the exploration capability and the levy flight shows the exploitation capability. According to the authors the moth search has weak exploitation ability therefore the differential evolution is embedded in moth search to aid the exploitation process. The population is divided in two halves in the MSDE approach, where one half is the representation of phototaxis and the other is of levy flight. The current solution is updated based on either the moth search algorithm or the differential evolution. The MSDE improves the makespan which is the fitness criteria in the technique but MSDE does not consider the load balancing of VMs. In [26] a vocalization behavior of humpback whale optimization algorithm (VWOA) was proposed. VWOA intended to improve the resource utilization, energy consumption, and execution cost. The objective function is composed of all the above-mentioned objectives. The algorithm is inspired by the whale mammal. It exhibits the behavior of mating of whales. The adult male whale sings in search of clusters. Then it joins other singing adult males. The secondary escort in the cluster competes with the major escort to win. The mathematical model is formulated with the distance and trigonometric relations for the defined behavior of whales. The half iterations assist the exploration while the other half exploitation. The VWOA does not consider the actual computational power of VM in the balancing of load, which stimulates the imbalance of resources.

Ivana in [71] proposed a hybridized monarch butterfly optimization algorithm for task scheduling. In this algorithm the population is considered to belong from two areas called as land 1 and land 2. The operations are performed to produce offspring which may be discarded or accepted based on the fitness as compared to their parents. The solution migration phase creates a new solution whereas the solution adjustment operator introduces both exploitation and exploration capabilities. To improve the original monarch algorithm a mechanism to discard

exhaustive solution through ABC metaheuristic is adopted. An additional parameter trial is introduced which increments in each iteration. When the solution is not improved for preset number of iterations, it is considered as exhausted and being replaced by random sequence. The control parameters of ABC are added to monarch to avoid the premature convergence and trapping in local minima. By choosing random sequence instead of discarded solution the algorithm may suffer slow convergence and it is highly possible that any local good point remains unexplored due to lack of exploitation. Pradeep in [75], proposed CGSA to improves the cost, energy consumption and memory usage. CGSA is a hybrid technique having combination of cuckoo search (CS) and gravitational search algorithm (GSA). The candidate solutions are updated first with GSA and then with CS. The best position of GSA or CS is replaced by the best fitness found between both GSA and CS. The performance is evaluated on a small dataset. The replacement of the best position disturbs the properties of one algorithm because best position particle participates in both techniques.

The Table 2.1 enlightens the strengths and vulnerabilities of some related techniques inferred from the study of researches. The key findings of literature study are also summarized through the listed research articles.

TABLE 2.1: State-of-the-art Scheduling Techniques

Techniques	**Strengths**	**Features**	**Weaknesses**
AIGA [8]	Improves makespan.	Adaptive probability of crossover and mutation is introduced.	1. Load is imbalanced.

ETA-GA [52]	Reduces probability of failure and completion time.	It has multi-objective optimization.	1. Converges pre-maturely due to over exploitation caused by the selection of best chromosomes to participate in population generation each time. 2. They have not considered load balancing.
MGGS [14]	Improves makespan and resource utilization.	It has Greedy Strategy to update vectors. Roulette wheel is used as selection operator.	1. Evaluated on a very small dataset. 2. Large number of genes of each chromosome are updated, therefore chromosome loses GA properties also it is computationally expensive.
RALBA [7]	Improves load balancing.	Share of each VM is calculated based on power and size of jobs.	1. Imbalance of resource utilization increases when number of jobs reduces.
TS-GA [19]	Improves makespan.	Tournament selection operation of GA is used.	1. Performance is not compared with other popular meta-heuristics. 2. Load is imbalanced.
DSOS [5]	Reduces makespan.	Fast Convergence.	1. Chances of getting trapped in local minima. 2. Load is not balanced among resources.

Multi-objective GA [32]	Provides load balancing and reduces makespan.	Two conflicting objectives are combined in a relation to define fitness as minimization function.	1. High probability of mutation which may lead to pre-mature convergence.
Hybrid PSO [57]	Improves makespan, resource utilization and convergence.	Velocity in PSO is updated using Differential Evolution algorithm.	1. Performance is not evaluated on any large-scale dataset.
Multi-objective PSO [82]	Improves time and cost of execution.	An archive of dominating and non-dominating particles is maintained.	1. Fittest particle being chosen does not meet both objectives because fitness function is not multi-objective. 2. Weak capability of exploration due to selection of fittest based on any of the objective between the multi-objectives.
Efficient GA [17]	Improves execution time of jobs.	LCFP and SCFP are used parallel to randomization for population initialization.	1. Load balancing is not considered in the objective function. 2. Performance is evaluated on a very small dataset.

2.3 Research Gap Analysis

In the literature there are many techniques [5, 8, 17, 19, 52] which have focused the execution time or makespan as an important parameter for improvement in task scheduling. Very few techniques have addressed the issue of considering load balancing [7, 14, 17, 32, 37, 57, 60, 67, 69, 76, 82] while not overlooking the need for improvement in makespan. However, the finishing time of tasks is a common measure opted as objective of scheduling and even for the evaluation of performance [5, 7, 14–19, 21, 22, 28, 29, 44, 56, 69, 76]. The heuristic schedulers do not possess the potential to explore the huge search space and therefore, meta-heuristics are used. The heuristic based techniques are being fused with meta-heuristics and this trend has been observed in the following researches [14, 17, 24]. In meta-heuristic based solutions, the problem of slow convergence encounters [52]. The convergence speed of meta-heuristic, exploration and exploitation capabilities are deemed in researches [10, 66, 73, 79] to meet the set objectives of optimization. The genetic algorithm has been extensively applied to schedule tasks [2, 8, 14, 17, 18, 23, 24, 44, 52, 54, 55]. Different improvements are suggested to improve the convergence of meta-heuristics [14, 20, 24, 25, 27, 34, 38, 52, 55, 57, 66, 71, 76] and GA based meta-heuristic [14, 24, 52, 55], which can be further enhanced. Some researches have also considered load balancing but actual information of resources, that is the consideration of power of VMs and size of jobs, is not focused. Techniques where load balancing is considered, the way in which the load is balanced matters a lot. Researchers have devised different ways of balancing the load but VMshare information is not deployed by any means in meta-heuristics and there is need to have application of it. To evaluate the performance of schedulers usually small synthetic datasets are deployed [14, 17, 23, 38, 40, 57, 72, 75, 85] which may not contribute to meticulously assess the performance [89].

Chapter 3

Methodology

This chapter begins with the recap of research problems highlighted in the introductory chapter. It describes the methodology having problem formulation and model of the presented solution. The description of datasets with necessary explanation is provided. The reasons and rationales to opt the presented solution have been described. Examples are provided where applicable to explain the steps involved in the methodology of presented techniques as well as to prove the statements. The motivation for the selection of algorithms and logical reasoning is emphasized, which are later backed by extensive experiments in the next chapter.

Below is the recap of research questions which are derived from the analysis of relevant literature. This chapter connects these research questions with the presented techniques.

1. Improving the makespan and load balancing.

2. Enhancing the convergence speed of GA.

3. Merging heuristic and meta-heuristic for getting the optimal task scheduling.

From the research questions the intention of this research is evident. The main concern is optimal task scheduling. The scheduling is optimal when it meets the QoS performance metrics for client and service provider. The two significant

performance measures under consideration are makespan and load balancing. The makespan and resource utilization are described in Section 3.7.

3.1 Multi-objective Optimization

Lot of techniques have been devised and proposed in the literature [2, 4, 5, 7, 8, 10, 11, 14–30, 32–34, 36–41, 44, 52–60, 65–67, 69, 71–88] for task scheduling in cloud computing environment. Most techniques address makespan but very few have concentrated on load balancing, whereas some have not considered both simultaneously and the impact of that is discussed later. There are variety of methodologies in the literature for improving these two measures of performance. In this research load balancing has been used as an important parameter for efficient task scheduling. The load balancing can improve makespan to some extent but relying only on the makespan cannot guarantee the optimal resource utilization. The example given below reveals the need for multi-objective optimization and the dependency of these two objectives. The unit for the representation of tasks is Million Instructions (mi) and for power of VM is Million Instructions Per Second (mips). The formulas used in the example are described later in this chapter.

Consider there are six heterogeneous tasks and three VMs. Although there are many possible mappings of these six tasks on three VMs. But consider few of them to analyze the behavior of makespan and resource utilization. One of the ways to compute the resource utilization is through the formula of ARUR and the same is used in the below illustration.

The Table 3.1 shows the specifications of tasks and the Table 3.2 shows the power specification of VMs used in the example. Table 3.3, 3.4, and 3.5 present solution 1,2 and 3 respectively. Each solution represents the tasks assigned to the respective VMs and these are three different mappings for the tasks and VMs specified in Table 3.1 and 3.2.

TABLE 3.1: Specifications of Tasks

Task no.	1	2	3	4	5	6
Size	100 mi	200 mi	300 mi	400 mi	500 mi	100 mi

TABLE 3.2: Power Specification of VMs

VM no.	1	2	3
Power	200 mips	300 mips	100 mips

TABLE 3.3: Solution no.1

Task no.	1	2	3	4	5	6
Assigned to VM no.	2	3	1	1	2	2

TABLE 3.4: Solution no.2

Task no.	1	2	3	4	5	6
Assigned to VM no.	2	2	3	1	2	1

TABLE 3.5: Solution no.3

Task no.	1	2	3	4	5	6
Assigned to VM no.	1	1	1	2	2	3

The Table 3.6 shows the calculations performed against the solutions for makespan and ARUR. The minimum value of makespan and the maximum value of ARUR are good. In the mentioned example the makespan of solution no. 2 and 3 is same but the ARUR is different. So, it means that solution no.2 has better resource utilization as compared to solution no.3, although the makespan is same. Among the three solutions the solution no.2 is best as the resource utilization is good and makespan is good too.

There can be many possible examples to cover all scenarios. The above solutions represent some of the possibilities. In general, the possibilities that may arise for the makespan and ARUR values are:

- Two different solutions may have same makespan but different ARUR values.

- Two different solutions may have same ARUR but different makespan values.
- Two different solutions may have same makespan and ARUR values.
- Two different solutions may have different makespan and different ARUR values.

The general perception is that when makespan is good ARUR is good and vice versa. But this is not always the case when two solutions are compared, the possibilities are mentioned above. It shows that depending on one factor like makespan or load balancing for achieving improvement in both is not as good as depending on both which can lead to much better outcomes. From the computation of makespan and ARUR of solutions it is inferred that depending on both makespan and load balancing is much beneficial. There is no point of finding a solution with good load balancing while having high makespan. Similarly having good makespan with imbalanced load is not good. Suppose the maximum load balancing achieved by any technique is 90% but there are different solutions where the load balancing is maximum. So, in such a case the best solution would be the one where makespan is minimum for 90% load balancing. This may also happen that the solution 'A' having good load balancing as compared to some other solutions B, C, D is not the solution with minimum makespan. The multi-objective fitness function is used in the presented work and the notion of relying on multi-objective fitness function is described in Section 3.3.3 and it is further enlightened there.

TABLE 3.6: Calculations of Makespan and ARUR

Solutions	**Completion Time Calculation**	**Makespan (seconds)**	**ARUR (0-1)**
Solution no. 1	$CT_{VM_1} = 700/200 = 3.5s$ $CT_{VM_2} = 700/300 = 2.33s$ $CT_{VM_3} = 200/100 = 2s$	3.5	0.74
Solution no. 2	$CT_{VM_1} = 500/200 = 2.5s$ $CT_{VM_2} = 800/300 = 2.66s$ $CT_{VM_3} = 300/100 = 3s$	3	0.9
Solution no. 3	$CT_{VM_1} = 600/200 = 3s$ $CT_{VM_2} = 900/300 = 3s$ $CT_{VM_3} = 100/100 = 1s$	3	0.77

The problem of scheduling is an optimization problem, therefore multi-objective optimization is required in which the use of meta-heuristics is common. The decision of using and choosing meta-heuristic is defined later.

3.2 Scheduling Approach

Before moving to the presented solution for achieving the multi-objective optimization in task scheduling, it is important to know where the presented solution lies in cloud environment. Scheduling can be at different levels as described earlier in Section 1.6, but here the focus is on task scheduling. The Figure 3.1 describes where the presented techniques fit in task scheduling problem of cloud computing. The working of the algorithms is described later.

The Figure 3.1 shows that the batch of tasks are received, and the necessary inputs are properly encoded before they are handed over to the schedulers. The cloud broker receives the mapping which is chosen by the scheduler to map the tasks to the available resources.

The meta-heuristic based solution can overcome the problems of poor load balancing and high makespan [2, 14, 24, 32]. Task scheduling is NP hard problem and it has many possible solutions but reaching to optimal solution is very challenging. The heuristic and meta-heuristic are described briefly in Section 1.4. Heuristics are mostly problem dependent, and they cannot have same behavior with different type of datasets [4, 30]. Meta-heuristics are stochastic in nature and each time the result may fluctuate to some degree, but they have consistent behavior of exploring the huge search space regardless of dataset [31]. They have been deployed successfully for multi-objective optimizations in the literature for task scheduling.

There are many meta-heuristics applied in task scheduling optimization as described in the literature review. The GA has good exploration capability [2, 8, 14, 17, 18, 23, 24, 44, 52, 54, 55] and it does not get trapped easily in local minima. It has lot of parameters and their fine tuning can lead to much better optimal

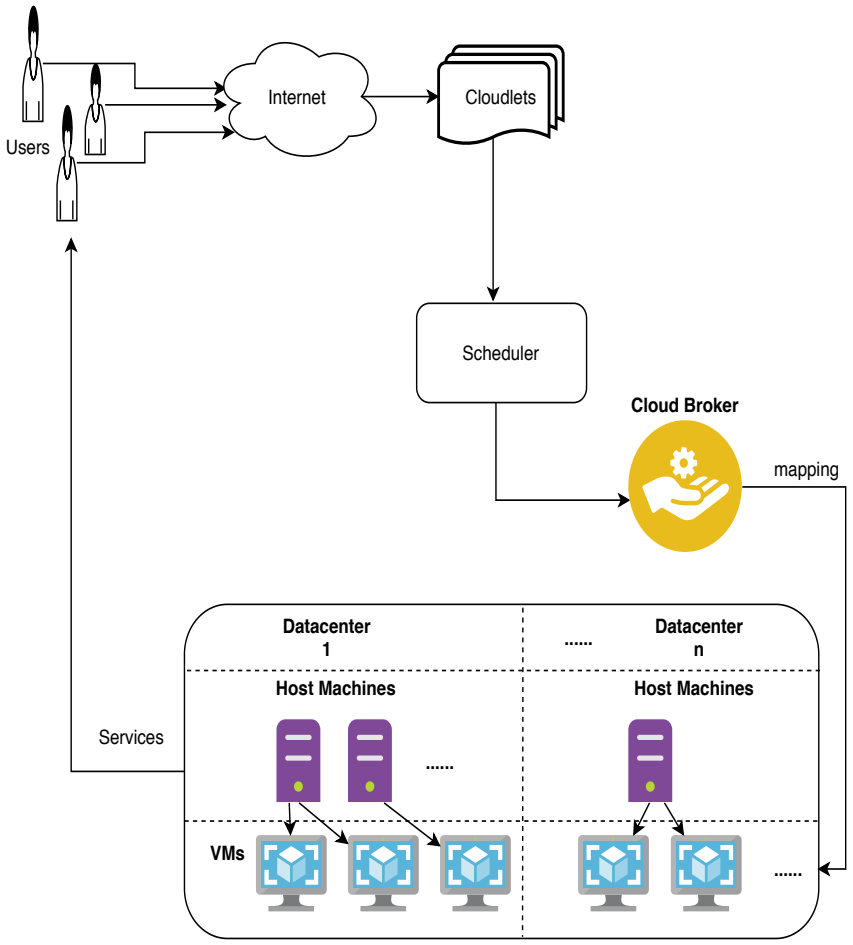


FIGURE 3.1: Schedulers in Cloud Environment

solution. GA has slow convergence but the probability to find global optima is high [90]. The presented solution provides two hybrid variants of genetic algorithm to address the research questions related to fast convergence, load balancing, and reducing makespan.

The optimization of task scheduling using Genetic Algorithm can improve makespan in a load balanced way [2, 14]. Makespan may improve without load balancing but the problem of under and overutilization of resources triggers. Therefore, load balancing should be one of the prime objectives for optimization.

The possible variants of GA can be following:

1. Hybrid Genetic Algorithm in which hybridization is through heuristic approach [14, 17, 18, 67, 91]. It has further three possibilities.

- One is to use heuristic to just initialize the population or as the beginning point of the GA [17, 18].
- Second is to encode the GA in such a way that it will hold the concept of heuristic in the representation of chromosomes [91].
- Third is to guide the optimization of GA by updating the chromosomes of GA using any heuristic [14].

The problem with using heuristic just for initialization is that the GA losses the properties of solutions being inculcated in the initial few generations of GA. After few iterations the new generation would have very least effect of the initial population because GA is not guided anymore with the heuristic. Encoding GA in such a way that it will hold the properties of heuristic is more about defining lot of rules to interpret and guide GA [91]. The third possibility of guiding GA with heuristic is lot more feasible that the actual algorithm keeps working and the convergence is guided through any good heuristic. The level of guidance may vary, and it should also be done in moderation to avoid over exploitation. The presented solution uses the hybridization of GA through heuristic to guide the solution. This version of GA is named as Balancer-GA.

2. There are many techniques proposed in the literature where GA is combined with other meta-heuristics [92, 93]. Hybridizing GA with other meta-heuristic is also possible but it must be for any specific reason. One reason to hybridize GA with any other meta-heuristic is to improve the convergence. Though GA has good optimization capabilities, but it suffers slow convergence [90]. To overcome this issue hybridization of GA with DSOS is proposed and this version of GA is named as Symbiotic-GA. The symbiotic organism search algorithm has good operations which exploit the solutions with reference to the best particle. It makes the convergence fast. Hence, SGA takes advantage of SOS while not compromising the benefits of GA.

3. Using pure GA is good but again the convergence is slow, also other parameters can be further improved by hybridization [2, 8]. The merger of heuristic is

additional advantage to the working of pure GA for task scheduling. Therefore, lot of solutions have been proposed so far to use Hybrid GA. This research also takes advantage of hybridization and provides two versions of GA for that.

In GA the probability of crossover and mutation plays a vital role in controlling the exploration and exploitation of the search space of candidate solutions [52]. The search of optimal solution needs to have a good balance between both exploration and exploitation. The adaptive probability is about making decision of the rate of probability depending on the current situation of population [8]. When probability of mutation is low initially, the chromosomes start converging towards better solution, and the moment they are trapped the probability is increased. The other possibility is to make probability of mutation high initially so the particles can explore more points and over the time the mutation rate is reduced so the algorithm can focus on any point and get converged. Throughout the journey of finding the optimal solution in GA the population suffers with different ups and downs. Therefore, defining probability requires extensive experimentation and algorithm should decide probability according to the state of the population. The standard deviation can define how far chromosomes are from each other based on the fitness value. When chromosomes are far based on fitness value means there is enough diversity and probability should reduce. Similarly, when chromosomes are close to each other the diversity should be enforced by introducing more randomness. The Equation 3.9 describes the formulation of SD for rate of probability. The probability rate of crossover defines how many particles will be reproduced for the subsequent generations. Survival is about elitism in which some solutions are moved as it is to the later generations so the good solutions may not lose, and it serves as a memory for GA. The crossover generates new chromosomes and some chromosomes copied from the previous generation are considered as elites. The significance of setting the opted mutation and crossover rate is described later.

3.3 Genetic Algorithm

3.3.1 Problem Encoding

The GA needs to have proper encoding of problem before compiling its evolutionary operators. There are two basic encoding of GA used in the literature for task scheduling problem. One is to use binary encoding of vectors and other is to use the discrete encoding. The chromosomes in GA represent the solutions and therefore, they should represent appropriate mapping of tasks to VMs.

The discrete encoding is also known as real number encoding. The chromosomes in discrete encoding are represented as vector of $1 \times n$, where n is the number of tasks. The vector representation is expressed in Table 3.8. Here each gene of chromosome represents the VM number where the corresponding task is mapped. It is not convenient to make vector of n such that n represents number of VMs because there can be more than one tasks mapped to one VM and in such way the constraints to avoid remapping of tasks need to be employed.

TABLE 3.7: Discrete Encoded Chromosome

Task no.1	Task no.2	Task no.3	...	Task no.n-2	Task no.n-1	Task no.n
VM	VM	VM	VM	VM	VM	VM

The other way to encode GA is using the binary scheme to represent mapping. There are two possible ways in the binary encoding. 1) The binary encoding has vectors of $1 \times n$ dimension [14]. Here n represents the number of tasks. In the mentioned scheme each gene is composed of multiple alleles which is a string of binary bits representing VM number. 2) In second way of binary representation the vector has $m \times n$ dimensions, where m represents number of VMs and n number of tasks as demonstrated in Table 3.8. The vector is multi-dimensional, and each row represents mapping of tasks to respective VM. The bit at any VM turns on while other remain off and it shows mapping of task to VM and there is no repetition. The later binary approach is less common in the literature.

Out of all described encoding schemes the discrete number encoding is the simplest and it has been used extensively in the literature. The discrete number encoding is used in both versions of presented GA. The GA begins with random population of n chromosomes and then the fitness of each chromosome is computed using the defined fitness function. In each generation based on some criteria the chromosomes are selected for reproduction operation. The new population of size n is generated using the crossover and mutation operations. The crossover is responsible to exploit the best chromosome in the search space and it also ensures the diversity to some extent. The mutation is responsible for the introduction of diversity in the population.

The initialization phase, selection operation, parameter tuning, fitness function, crossover and mutation opted for the presented techniques are defined below and these are same for both versions of presented techniques. The specific differences of both version with their benefits in achieving the solution are defined later in this chapter.

3.3.2 Initialization

There are many possible ways to initialize the population of GA. The main concern in this phase is to ensure enough diversity in the population such that there would be no over exploitation in the consequent iterations of GA. Therefore, usually GA is initialized with random values. As the presented technique encodes GA with discrete values in which the value represents the VM number that is why population of 'n' chromosomes is generated. The values of genes in chromosomes depend on the number of VMs in the dataset specification. Let us say, there are three VMs and six tasks using the same example mentioned in Section 3.1. The initialization phase generates random values in between 1 and maximum number of VM. Let 'A' is a randomly generated chromosome A= {1,2,1,2,1,1}.

The term job or task can be used interchangeably. The above vector is a chromosome which shows that the first job is mapped to VM number 1, the second job

is mapped to VM number 2 and so on. Each value is a VM number and in terms of GA it is known as gene of chromosome. The value of gene cannot exceed 3 because there are only three VMs in the example under consideration. The above chromosome shows that there is no job mapped to VM number 3. On the other hand, the VM number 1 has 4 jobs. Regardless the power of VM or the length of job the initialization phase builds the chromosomes, each representing a mapping scheme. The number of chromosomes is known as the population size and it depends on the problem and need. In the presented technique the population of 120 chromosomes is used to ensure that huge possibilities of solutions can be generated and manipulated later. Hence it is helpful in exploring the huge search space. When population size is too large the GA takes lot of time, but it introduces more diversity in the population. On the other hand, when population size is too small it may force the GA to converge prematurely because the diversity in the population is compromised. With too small population size the computations does not take much time but lot of iterations are required to get a better solution which may not be so good. So, there should be a balance between the too small and large population size, and it depends on the problem. Normally, the population size is set with the experimentation and in the presented technique it is set to 120.

3.3.3 Fitness Function

The fitness of every chromosome is a measure to evaluate the worth of any solution and each chromosome is a solution just like solutions presented in example in Section 3.1. The solution in this problem defines a mapping of tasks to VMs. The fitness function can be single or multi- objective. In the presented technique the fitness is based on multi-objectives namely the makespan and load balancing. In the literature there are many different ways to balance the load and to calculate the resource utilization. Calculating the resource utilization in the fitness function can be further improved by using the information of percentage of VM share used. Experimentally and logically it gives much better idea to evaluate a solution

through such fitness value and hence the workload can be balanced knowing the right information. The composition of fitness function is based on both makespan and load balancing information and it is essential because according to set objectives the makespan should improve but in load balanced way. The computation of fitness function is presented in algorithm 1.

Makespan is the finishing time of batch of jobs. The jobs are mapped on different VMs with different computational powers. Therefore, the makespan value depends on the completion time of assigned jobs on any VM. The VMs are working in parallel that is why the makespan value is the maximum completion time of running VMs. The makespan is calculated using the formula in Equation 3.12 and 3.13, and sample calculation is demonstrated using the chromosome represented in example shown above.

Using the above-mentioned equations, the completion time of VM number 'j' is computed by adding the job size in million instructions of jobs number 'i' that are mapped to VM number 'j'. The sum of job size is divided by the total power of that VM. The Map[i,j] is a variable of Boolean type whose values depend on the mapping of job number 'i' to VM number 'j'. The Map[i,j] vector is exemplified in Table 3.8.

TABLE 3.8: Binary Encoded Chromosome

	Job1	Job2	Job3	Job4	Job5	Job6
VM1	0	0	1	1	0	0
VM2	1	0	0	0	1	1
VM3	0	1	0	0	0	0

The power of VMs and Job sizes are described already in example in Section 3.1. To find the makespan of solution number 1 in example, first the completion time of all three VMs is calculated. The computation is shown:

$$CT_{VM_1} = \frac{100\times0+200\times0+300\times1+400\times1+500\times0+100\times0}{200} = \frac{700}{200} = 3.5$$

$$CT_{VM_2} = \frac{100\times1+200\times0+300\times0+400\times0+500\times1+100\times1}{300} = \frac{700}{300} = 2.33$$

$$CT_{VM_3} = \frac{100\times0+200\times1+300\times0+400\times0+500\times0+100\times0}{100} = \frac{200}{100} = 2$$

$$makespan = \max(3.5,\ 2.33,\ 2) = 3.5$$

Makespan is one of the objectives in both version of the GA. The second objective is load-balancing and the formula of finding load balancing is formulated with the notion of using the percentage of share used.

The value of load balancing is calculated with the help of Equations 3.1 to 3.7. It describes the total VM share used based on the computational power of VM and the total size of batch of jobs in million instructions.

$$ShareRatio_{vm_j} = \frac{VM_j.mips}{\sum_{j=1}^{n} VM_j.mips} \tag{3.1}$$

The $ShareRatio_{vm_j}$ is used to find the percentage of share of each VM, depending on the power of VM. When there are 'n' number of VMs then the load of each VM depends on the total available power of the VMs to make balance of load among VMs. Therefore, the power of each VM in million instructions per second is divided by the total power of VMs means the total million instructions per second that is the available power of VMs. The value of $ShareRatio_{vm_j}$ is between $0 <= \ ShareRatio_{vm_j} <= 1$, where '0' represents no workload and '1' represents 100 percent workload that should be mapped to any VM. In this way the amount of workload to be mapped for perfect balance is assessed and mapping can be defined such that load would not exceed or fall short.

$$VMShare_j = (\sum_{i=1}^{m} cloudlets) \times ShareRatio_{vmj} \tag{3.2}$$

To compute the exact amount of million instructions that should be mapped on any VM, the size of jobs available in the batch are added and then multiplied by the $ShareRatio_{vmj}$. This provides the total size of jobs in million instructions to be mapped on VM_j. In this the VM having any percentage of power share is

assessed to have no more million instructions than the computed share for that VM.

The share used by any VM is calculated against the chromosomes of GA using the Equations 3.3 to 3.7. The share used is based on the million instructions mapped on the VM in the form of jobs. The size of all jobs 'i' is summed such that job 'i' is mapped to VM 'j' to find the share used by that VM.

$$ShareUsed_j = \sum_{i=1}^{n} (jobsize_i \times Map[i,j]) \tag{3.3}$$

$$PSU_j = \frac{ShareUsed_j}{VMShare_j} \times 100 \tag{3.4}$$

The percentage of share used (PSU) is defined to be a measure which is computed against chromosomes generated in GA. It provides the information about how much percentage of share is used with the mapping specified by the chromosome. When PSU is less than the share of any VM, the value of PSU would be less than 100 and it means the VM is underutilized. Similarly, when PSU is greater than 100 means the VM is overutilized. The Equation no. 3.5 and 3.6 are used to scale the PSU value between 0 to 1.

$$if\ PSU_j \leq 100,\ \frac{PSU_j}{100} \tag{3.5}$$

$$if\ PSU_j > 100,\ \frac{100 - (PSU_j - 100)}{100} \tag{3.6}$$

$$Avg_{PSU} = 1 - \frac{\sum_{j=1}^{n} PSU_j}{n} \tag{3.7}$$

The average PSU in Equation no. 3.7 is the value of load balancing factor and it is between 0 and 1, where 1 represents the minimum load balance and 0 represents the maximum load balance. The objective function in both versions of presented

GA utilizes the value of makespan and load balancing. The objective function is defined to be a minimization function having two different factors where makespan is a minimization function, and load balancing is maximization function. Hence, the maximization function of load balancing is transformed to minimization by subtraction from 1, to make it consistent with the minimization function.

The two objectives are combined in a relation which is calculated using the Equation 3.8 as a minimization function. The objective function is named as load_balancer as it is contributing in the balancing of load while also considering the minimization of makespan. There may be many possible solutions where load balancing is same but makespan is different as already discussed in the possibilities defined in Section 3.1. On improving the load balancing it is possible that makespan does not change or even increase. To effectively optimize multi-objectives both makespan and ARUR are considered and combined through addition. Addition represents that a good solution would be the one having minimum makespan and for that minimum makespan value the load balancing should be good. For example, if the fitness value is 30.5 and 30.2, the solution having 30.2 fitness value is better. If two solutions have fitness values 30.1 and 29.2 then 30.1 would not be considered as the load balancing is good but makespan is not good. So, the fitness function will give priority to the solution producing minimum makespan and good load balancing for that makespan. Load balancing defines that how much size of tasks should be mapped on any VM but which task to be mapped is not defined by the load balancing. The GA is used because of huge search space of task scheduling problem and there are many possible solutions, but GA works on the solutions equal to the size of population to find optimal solution. The chromosomes of GA represent mapping where each VM gets a pool of tasks. Load balancing in the fitness function is achieved through considering the VM share of each VM and the pool of tasks assigned to any VM should not exceed the limit of share of each VM. It is still possible that even with good load balancing the makespan may not be good as compared to any other solution or chromosome. Logically there is no advantage of load balancing when tasks are taking much time in execution. Here it is to be noticed that the selection of solution with good makespan and

then considering the load can ultimately lead to a good solution with better load balancing as well.

$$load\_balancer\,(x) = makespan + Avg_{PSU} \tag{3.8}$$

The calculation of makespan is already demonstrated with an example earlier and the computation of Avg_{PSU} for the solution no.1 is explained below using the same details mentioned in example in Section 3.1.

In example there are three VMs so ShareRatio computation for all three VMs is as follows:

$$ShareRatio_{vm_1} = \frac{200}{200+300+100} = \frac{200}{600} = 0.34$$

$$ShareRatio_{vm_2} = \frac{300}{200+300+100} = \frac{300}{600} = 0.5$$

$$ShareRatio_{vm_3} = \frac{100}{200+300+100} = \frac{100}{600} = 0.16$$

The amount of workload that each VM should have is given by $VMShare_j$ and it is computed as follows:

$$VMShare_1 = (100 + 200 + 300 + 400 + 500 + 100) \times 0.34 = 544$$

$$VMShare_2 = (100 + 200 + 300 + 400 + 500 + 100) \times 0.5 = 800$$

$$VMShare_3 = (100 + 200 + 300 + 400 + 500 + 100) \times 0.16 = 256$$

Adding all three VMShare values we get 1600 which is the total size of batch of jobs. It is evident that according to the power of each VM the workload that VM should get for perfect balance is the value of VMShare. The mapping in form of chromosomes generated using GA is evaluated for fitness value using the above shown mechanism of computing the overload and underload. From the mapping defined in solution no.1 of the example the PSU value for each VM is calculated.

$$ShareUsed_1 = 100 \times 0 + 200 \times 0 + 300 \times 1 + 400 \times 1 + 500 \times 0 + 100 \times 0 = 700$$

$$ShareUsed_2 = 100 \times 1 + 200 \times 0 + 300 \times 0 + 400 \times 0 + 500 \times 1 + 100 \times 1 = 700$$

$$ShareUsed_3 = 100 \times 0 + 200 \times 1 + 300 \times 0 + 400 \times 0 + 500 \times 0 + 100 \times 0 = 200$$

The VM1 in example handles 700 (mi) which is more than its share, where VM2 and VM3 fall short in utilization of share for mapping defined by solution no. 1. The computation of PSU is expressed below.

$$PSU_1 = \frac{700}{544} \times 100 = 128.67$$

$$PSU_2 = \frac{700}{800} \times 100 = 87.5$$

$$PSU_3 = \frac{200}{256} \times 100 = 78.125$$

After scaling using Equation 3.5 and 3.6 the value of PSU is as follows:

$$PSU_1 = 0.713$$

$$PSU_2 = 0.875$$

$$PSU_3 = 0.781$$

The PSU value for all three VM is scaled between 0 to 1. Both over and under-utilization is converted to a same scale.

$$Avg_{PSU} = 1 - \frac{0.713+0.875+0.781}{3} = 1 - \frac{2.369}{3} = 1 - 0.78 = 0.22$$

The Avg_{PSU} is converted to minimization value in which 0 is best and 1 is worst. After combining both values in a relation of objective function the final fitness value of solution no.1 is as shown below.

$$load\_balancer(x) = 3.5 + 0.22 = 3.72$$

The objective function with multiple objectives can be combined in number of ways. Combining makespan and Avg_{PSU} factors which are both minimization function in an addition relation, gives enough information for multi-objective optimization. Over the iterations in GA the chromosomes are updated and makespan value is reduced. At a certain stage the makespan value no longer improves but due to the load balancing value in the objective function it is possible to select the best chromosome with minimum makespan and maximum resource utilization.

Algorithm 1 Fitness

Input: $Population[x_i]$
Output: $load\_balancer[x_i]$

procedure FitnessAlgorithm
  $SizePopulation \leftarrow size(Population[\ ])$
  $JOBsum \leftarrow 0$
  $VMsum \leftarrow 0$
  $AvgPSU \leftarrow 0$
  for $i \leftarrow 1$ *to* $Job[\ ]$ **do**
    $JOBsum \leftarrow JOBsum + Job[x_i]$
  end for
  for $i \leftarrow 1$ *to* $VM[\ ]$ **do**
    $VMsum \leftarrow VMsum + VM[x_i]$
  end for
  for $i \leftarrow 1$ *to* $VM[\ ]$ **do**
    $VMShare \leftarrow VM[x_i]/VMsum \times JOBsum$
  end for
  for $i \leftarrow 1$ *to* $SizePopulation$ **do**
    $VMmakespan[\ ] \leftarrow 0$
    for $j \leftarrow 1$ *to* $Gene[\ ]$ **do**
      $VMmakespan[Gene[x_j]] \leftarrow VMmakespan[Gene[x_j]] + jobsize[x_j]$
    end for
    for $k \leftarrow 1$ *to* $VM[\ ]$ **do**
      $value \leftarrow VMmakespan[k]/VMShare[k] \times 100$
      if $value <= 100$ **then**
        $PSU[k] \leftarrow value/100$
      else
        $PSU[k] \leftarrow (100 - (value - 100))/100$
      end if
      $AvgPSU \leftarrow AvgPSU + PSU[k]$
      $VMmakespan[k] \leftarrow VMmakespan[k]/Power\_of\_VM[k]$
    end for
    $AvgPSU \leftarrow 1 - AvgPSU$
    $load\_balancer[x_i] \leftarrow Max(VMmakespan[\ ]) + AvgPSU$
  end for
end procedure

3.3.4 Selection Operation

In GA the selection of best chromosomes leads to improvement in the fitness value. There are many possible strategies for the selection of chromosomes. The selected chromosomes participate in the evolutionary reproduction operations which are described later. The selection of parents is based on the fitness value obtained from the fitness function. In both versions of GA described later, the fitness is a minimization function. Therefore, the chromosomes having the minimum value can be considered as the best. It is not always good to select chromosome with the best fitness value because it can lead to over exploitation. The selection operator for the selection of parents in GA is responsible for selection such that the diversity is maintained, and population does not converge prematurely. In the literature [2, 8, 14, 17, 18, 23, 24, 44, 52, 54, 55] there are different selection operators used in GA and each have their own strengths and weaknesses. But they are chosen according to the problem and their implementation also vary. One of the operators that has been used to ensure diversity in the population is the proportionate selection operator. The mechanism of proportionate selection and rationale for selection are described below.

The proportionate selection operator exhibits the behavior of a roulette wheel spin. The proportion of fitness value is considered to have the probability of selection of chromosomes. The chromosomes having good fitness value have high proportion and therefore they can be selected more than other chromosomes for mating. In this way the fittest chromosome is not always selected, and other chromosomes can also get the chance. It is possible that the selection of other chromosomes which are not best, may introduce more diversity in the population. On the other hand, the selection of any best chromosome each time for some generations may lead to over exploitation in population.

The fitness value of chromosomes can be scaled to fit in a circular wheel. The wheel is circular and has divisions of 'n' pies. Each pie represents one chromosome of the entire population. The size of the pie is proportional to the fitness of the chromosome. The chromosome with high fitness has larger pie. There is a fixed

point 'p' directing on the roulette wheel. The roulette wheel is rotated two times for the selection of two chromosomes to participate as parents in the reproduction operator. In second spin a different chromosome is selected. The region pointed by the fixed point 'p' after the spinning of roulette wheel stops, is a pie representing one of the chromosomes of the entire population. More the size of pie of any chromosome, the more are the chances of selection of that chromosome.

The fitness values of all three solutions is given in the Table 3.9 and they are computed using the formula of load balancer.

TABLE 3.9: Fitness Values of Sample Solutions

Solutions	Load Balancer	Scaled
1	3.72	$10 - 3.72 = 6.28$
2	3.09	$10 - 3.09 = 6.91$
3	3.29	$10 - 3.29 = 6.71$

As the fitness function is a minimization function therefore the second solution is the best one. But to give much proportion of wheel to it the value is converted to maximization function for only selection operation. The best chromosome or a solution has more chances of selection in each generation among the entire population. But roulette wheel ensure that other chromosomes would also get the chance with the probability equal to their proportion.

From implementation point of view the proportionate selection is executed by following the steps mentioned below:

1. Converting fitness value to maximization value by subtracting from any large value greater than the maximum value.
2. Sorting the chromosomes according the fitness value.
3. Adding the fitness value to get the sum 's'.
4. Generating a positive random number 'r' whose maximum limit is the sum 's'.
5. Adding in 'r' the fitness values of chromosomes selected one by one in descending order.

One modification proposed for the proportionate selection in both version of GA is to ignore the extremely worst chromosomes. The worst chromosomes are ignored by the following steps.

1. Adding the fitness values to get the sum 's'.
2. Converting fitness value to maximization value by subtracting from any large value greater than the maximum value.
3. Sorting the chromosomes according the fitness value.
4. Generating a positive random number 'r' whose maximum limit is the sum 's'.
5. Adding in 'r' the fitness values of chromosomes selected one by one in descending order.

Using the above steps, based on the fitness value the worst chromosomes do not get any chance of selection and the best or average chromosomes are selected depending on their probability of proportion. The roulette wheel selection operation is presented in algorithm 2, and it is used in both versions of presented techniques.

Algorithm 2 Selection

Input: $load\_balancer[\,]$
Output: $Population[x_i]$

1: **procedure** SELECTION
2: $SizePopulation \leftarrow size(Population[\,])$
3: $S \leftarrow sum(load\_balancer[\,])$
4: $r \leftarrow random(S)$
5: $load\_balancer[\,] \leftarrow scale(load\_balancer[\,])$
6: $load\_balancer[\,] \leftarrow sort(load\_balancer[\,])$
7: $j \leftarrow 1$
8: **while** $r < S$ **do**
9: $r \leftarrow r + load\_balancer[x_j]$
10: $j \leftarrow j + 1$
11: **end while**
12: **return** $Population[x_j]$
13: **end procedure**

3.3.5 Crossover

The crossover is a reproduction operator of GA. It is used to produce two new chromosomes which are the result of crossover of parent chromosomes. The crossover

operator generates the number of chromosomes depending on the population size. The probability of crossover defines the number of chromosomes that are produced using the crossover operation. The rest of the population is filled with the best chromosomes of the previous generation. The best chromosomes that reserve their slot in the next generation are known as elites and the concept is known as elitism. The elitism is necessary to make certain the memory of best chromosomes found so far over the iterations in GA. Otherwise the best chromosomes may lose, and the performance of GA would degrade. The crossover probability is a decision factor in GA and the value of probability in both version of presented GA is defined later. The offspring produced with the crossover operation possess the properties of the parents. The parents are selected using the selection operator and then two offspring are produced from the parents by exchanging the genes of parent chromosomes. The diagrammatic representation is shown in Figure 3.2 and 3.3.

There are many different types of crossover operators among them two are very popular. These two crossover operators are single and multi-point crossover and one can also have the combination of both. There are different ways of application of single and multi-point crossover operators and one of them is to exchange the genes of chromosomes. In the presented technique both single and multi-point crossover is used to produce the offspring for generating the new population. Some are produced with single point crossover and some with multipoint to take advantage of both ways. The multi-point crossover introduces more diversity over the single point crossover. To have more diversity in the population the presented technique generates high proportion of chromosomes with multi-point and less with single point crossover. The percentage of offspring generated using single and multipoint crossover is defined later and the percentage is chosen with experimentation. The cut point in the parents is chosen randomly and it can be single or multi-point.

In single point crossover, one cut-point 'c' is selected in both parents such that $1 < c < size$ and two new chromosomes are produced. If chromosome 2 and 3 of example in Section 3.1 are selected as parents with random cut point $c = 3$, then offspring generation is as exhibited in Figure 3.2.

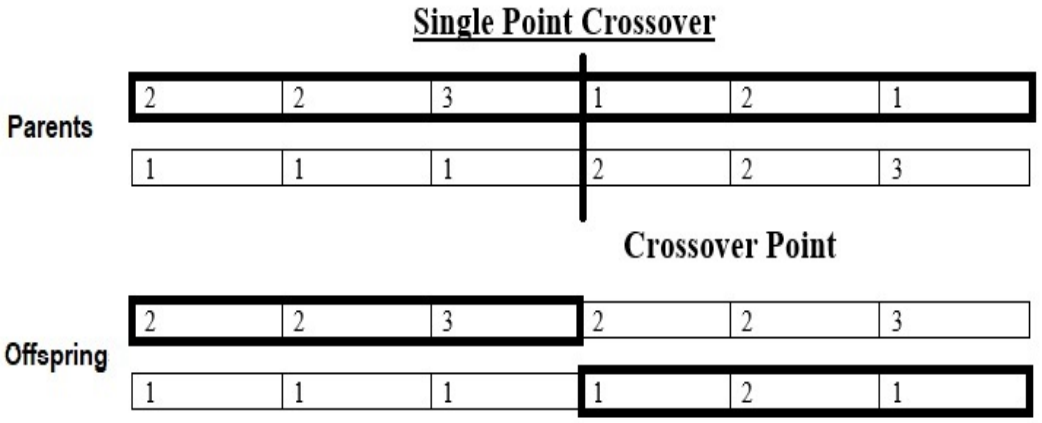


FIGURE 3.2: Single Point Crossover

In multi-point crossover, two cut-points 'c1' and 'c2' are selected in both parents such that $1 < c1 < (size - 1)$ and $c1 < c2 < size$. If chromosome 2 and 3 of example in Section 3.1 are selected as parents with random cut points $c1 = 2$ and $c2 = 4$, then offspring generation is as shown in Figure 3.3.

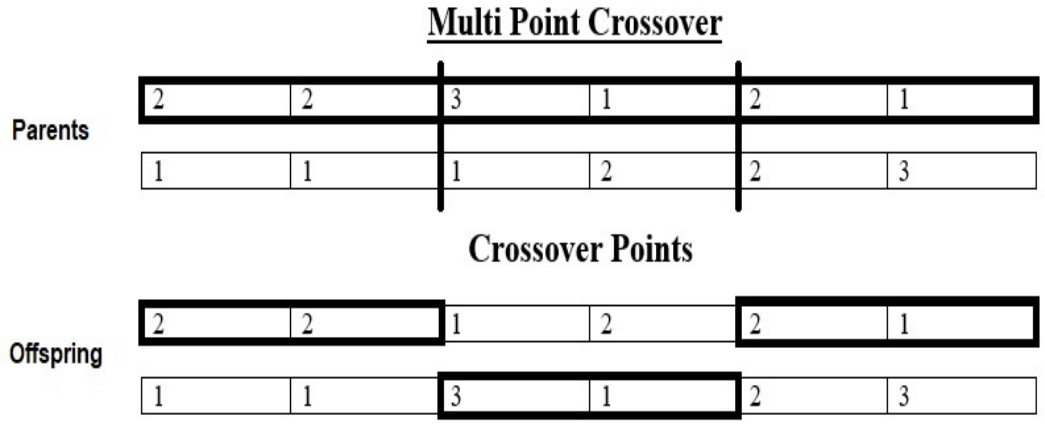


FIGURE 3.3: Multi Point Crossover

The newly formed chromosomes have genes which are exchanged from the parents. In this way the GA exploits the previously formed solutions and it also ensure some exploration. But to have significant exploration where no gene of the parent is adopted, the mutation operator is used. The crossover operator of presented technique is presented in algorithm 3.

Algorithm 3 Crossover

Input: $Population[Parent_1], Population[Parent_2], flag, p$
Output: $Population[x_i], Population[x_{i+1}]$

```
1: procedure CROSSOVER
2:     SizeChromosome ← size(Population[x_i])
3:     CutPoint_1 ← random(SizeChromosome)
4:     CutPoint_2 ← random(SizeChromosome)
5:     if flag == 1 then
6:         for j ← 1 to Cutpoint_1 do
7:             Child_1[x_j] ← Population[Parent_1]
8:             Child_2[x_j] ← Population[Parent_2]
9:         end for
10:        for j ← CutPoint_1 + 1 to SizeChromosome do
11:            Child_1[x_j] ← Population[Parent_2]
12:            Child_2[x_j] ← Population[Parent_1]
13:        end for
14:    else
15:        for j ← 1 to Cutpoint_1 do
16:            Child_1[x_j] ← Population[Parent_1]
17:            Child_2[x_j] ← Population[Parent_2]
18:        end for
19:        for j ← CutPoint_1 + 1 to CutPoint_2 do
20:            Child_1[x_j] ← Population[Parent_2]
21:            Child_2[x_j] ← Population[Parent_1]
22:        end for
23:        for j ← CutPoint_2 + 1 to SizeChromosome do
24:            Child_1[x_j] ← Population[Parent_1]
25:            Child_2[x_j] ← Population[Parent_2]
26:        end for
27:    end if
28:    Population[p] ← Child_1[ ]
29:    p ← p + 1
30:    Population[p] ← Child_2[ ]
31: end procedure
```

3.3.6 Mutation

The mutation operator is applied in GA on the chromosomes formed by the crossover operation. The notion behind mutation as the name implies is to mutate the genes. The mutation operator brings diversity in the population by introducing new genes in the chromosome. If there is only crossover operator, then the genes would be no different from the previous generation. The introduction of new gene may not contribute in improvement in fitness in case of only crossover, but

through mutation diversity is introduced in the population. From task scheduling point of view, there may be a VM to which no or very few jobs are assigned in any generation, so in the subsequent generations the mutation operator raises the probability of mapping of jobs to that VM.

Usually the mutation operation is not applied on the elites and rest of the population undergoes the mutation with the defined mutation probability. There are different types of mutation operators like gene swapping, random gene generation and so on. The random gene generation is known as random resetting mutation operator. The mutation operator used in the presented technique is random resetting and it generates random VM number against any job in the chromosome based on the probability. The detailed computation of mutation for the presented technique is exhibited using the chromosomes defined later in example in Section 3.1. The mutation probability states the probability of mutation of genes in a chromosome. There are two ways to define the mutation probability which are as follows:

1. One way is that the number of genes or bits (in case of binary), to be mutated, depends on the probability of mutation. For instance, the probability of 70% means that 70% genes of a chromosome would be mutated and 30% would remain same.

2. Another way is that mutation of each gene or bit (in case of binary) of a chromosome depends on the value of probability.

In the presented technique the probability of mutation is applied on each gene and the mutation rate or probability of mutation is defined later. Usually the probability of mutation is chosen with experimentation and it is a key factor to improve the global search capabilities of GA. There are different approaches regarding the selection of mutation rate, but it depends on the problem. The mutation rate can be fixed value that remain constant throughout the generations of GA. Mutation rate can be variable that changes in the course of GA. The variable mutation rate is called adaptive when it changes automatically with some defined constraints.

When mutation rate is too small then the algorithm would take lot of time to converge. Low mutation rate may cause premature convergence as the exploitation supersede the exploration of search space. On the other hand, when mutation rate is too high the chances of skipping any optimal local minimum, which could be global minima, increases and the exploitation is compromised. The exploitation is necessary in the search process of global optima to exploit the nearest points of global optimum. Therefore, there should be a balance in mutation rate to avoid situation of premature or failure of convergence. One approach is to set mutation rate very high at beginning of GA so that the GA can explore lot of points in the huge search space. Later when GA finds some better chromosomes the mutation rate is decreased to help GA focus on the optimal points found so far by adding more exploitation capabilities. Another approach is to set mutation rate very low in beginning so GA would not skip any local optimal point and later when GA starts to entrap in local optima the mutation rate is increased to jump pass the local optimal point in order to avoid premature convergence. The mutation rate chosen in the presented technique is based on the extensive experimentation discussed in the next chapter and also the standard deviation of fitness value is evaluated to make sure that there is enough diversity in the population or not. The equation of standard deviation is presented in Equation 3.9.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(f\,(x)_i - f\,(x)\,mean)^2}{size\ of\ pop}} \tag{3.9}$$

The example of mutation is demonstrated below using the chromosome presented in example in Section 3.1. Suppose the mutation rate 'm' is 0.02 and a random number generated are given in the Table 3.10. Here 0.02 represents the probability of 2%. The probability is checked against each gene through a random number 'r' between 0 and 1. If $r <= m$ then a random number 'n' is generated such that 'n' is less than number of VMs, otherwise the value of 'n' is equal to the previous value of the gene of chromosome. The solution no. 3 of example in Section 3.1 is updated using the values of 'n' as shown.

TABLE 3.10: Random Mutation Example

Value of r	**Solution no.3**	**If** $r <= m$	**N**
0.5	1	False	1
0.7	1	False	1
0.005	1	True	3
0.006	2	True	1
0.12	2	False	2
0.34	3	False	3

We can see the difference after mutation operation applied on solution no. 3 in the Figure 3.4.

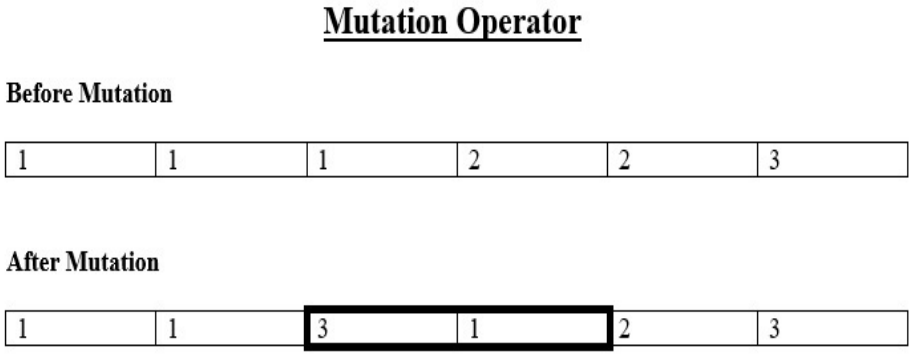


FIGURE 3.4: Mutation Operation

The 3^{rd} and 4^{th} genes of the chromosome are replaced with a random number and this changes the chromosome. The newly formed chromosome has some properties which are passed from the genomes of parents and two new genes are added which may not be totally different from the parents. Without mutation the crossover cannot bring significant diversity therefore mutation operation is essential. The mutation operator is presented in algorithm 4.

Algorithm 4 Mutation

Input: $Population[x_i]$
Output: $Population[x_i]$

```
1: procedure MUTATION
2:     SizeChromosome ← size(Population[x_i])
3:     for i ← 1 to SizeChromosome do
4:         r ← random(1)
5:         if r <= probability then
6:             r ← random(SizeChromosome)
7:             Gene[x_i] ← r
8:         end if
9:     end for
10: end procedure
```

3.4 BGA

The balancer GA is presented to balance the workload by updating the chromosomes of GA with a heuristic approach. From the literature it has been observed that guiding GA with heuristic can have positive impact on the fitness value to meet goal of global convergence [14]. The benefits of meta-heuristics and selection of meta-heuristics over heuristic has been discussed already. The merger of heuristic approach and meta-heuristic has shown improvement in fitness value as revealed in [14]. The presented balancer GA shows improvement over other popular state-of-the-art techniques and it has been proved experimentally in the next chapter.

The BGA follows the same steps of GA having all properties and evolutionary operations of GA. The initialization, elitism, selection, crossover, mutation and other necessary steps of GA in the presented technique are already discussed in detail. This section briefs the BGA technique and the necessary relevant discussion of the presented technique.

The aim of balancer GA is to take in consideration the over and under-utilization of virtual machines. The mapping defined by the chromosomes of GA can be further enhanced by balancing the workload and the balancer updates some genes of chromosomes. There are some researches which suggested that initialization phase should have heuristic, some have applied heuristics to update large proportion of

chromosomes and some have updated partial chromosomes or few chromosomes. They are discussed earlier in this chapter, but this section describes balancer GA which updates partial chromosomes. The detail of BGA reveals the benefits of updating partial chromosome with the presented technique.

The flow chart of BGA is shown in Figure 3.5, and it shows that balancer operator is proposed side by side to the regular evolutionary operators. The algorithm begins with the initialization phase and the population passes through the crossover and mutation phase following the parameter settings briefed earlier. After the mutation phase the balancer algorithm is called to balance the chromosomes based on the VM share, just like the VM share computed in the fitness function.

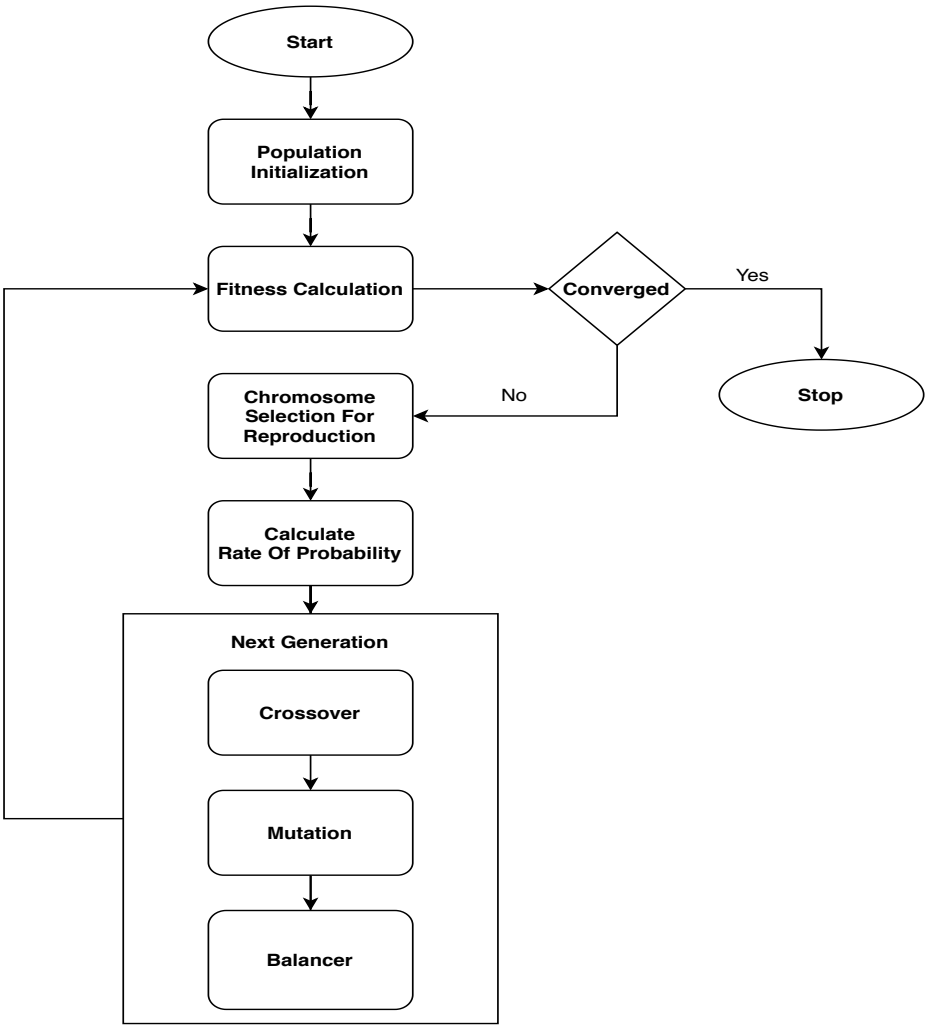


FIGURE 3.5: Balancer Genetic Algorithm

The balancer genetic algorithm is presented in the algorithm 5. The fitness, selection, crossover and mutation operations are briefed in algorithm 1,2,3, and 4 respectively.

3.4.1 Balancer Algorithm

The balancer is a presented heuristic being used in GA to update the chromosomes with the concept that the overloaded and underloaded VMs should reconsider the mapping of jobs. The Figure 3.6 shows the flow of balancer algorithm.

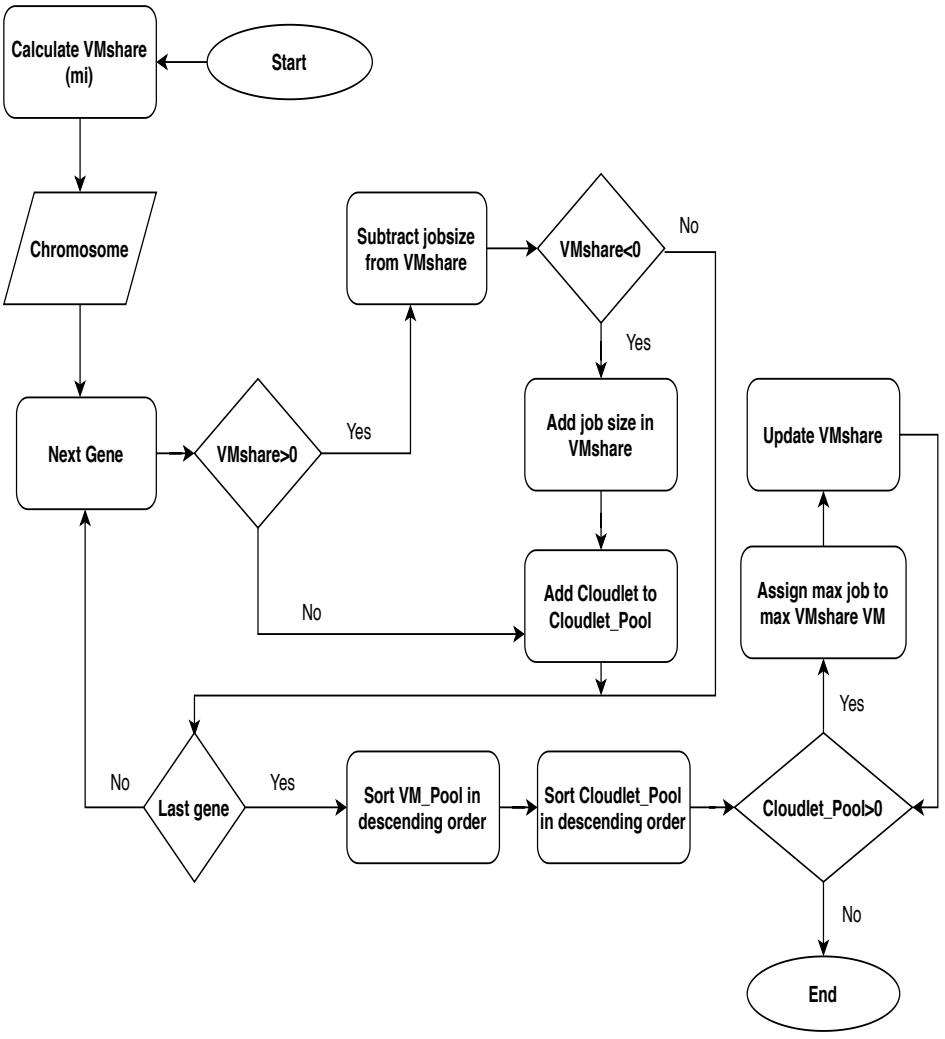


FIGURE 3.6: Balancer Operation

The VM share of each VM is calculated using the formula of VM share defined earlier. The chromosome is iterated gene by gene and the load on VM is computed

Algorithm 5 Balancer GA

Input: Size of Jobs, Power of VMs
Output: Mapping of Jobs to VMs

$Population[\,] \leftarrow random(Num\_of\_VMs)$
for $i \leftarrow 0$ *to* n **do**
  $load\_balancer[\,] \leftarrow FitnessAlgorithm(Population[x_i])$
end for
$SizePopulation \leftarrow size(Population[\,])$
while GA does not converge **do**
  $newPopulation \leftarrow 0$
  $p \leftarrow 3$
  $s \leftarrow 1$
  $Parent_1 \leftarrow 1$
  $Parent_2 \leftarrow 1$
  $s \leftarrow (SizePopulation/2) - 2$
  $SinglePoint \leftarrow s \times 0.2$
  $MultiPoint \leftarrow s \times 0.8$
  while $newPopulation < s$ **do**
    while $Parent_1 == Parent_2$ **do**
      $Parent_1 \leftarrow SelectionAlgorithm(load\_balancer[\,])$
      $Parent_2 \leftarrow SelectionAlgorithm(load\_balancer[\,])$
    end while
    if $SinglePoint > 1$ **then**
      $flag \leftarrow 1$
      $Population[\,] \leftarrow CrossoverAlgorithm(Population[Parent_1], Population[Parent_2], flag, p)$
    else
      $flag \leftarrow 2$
      $Population[\,] \leftarrow CrossoverAlgorithm(Population[Parent_1], Population[Parent_2], flag, p)$
    end if
    $SinglePoint \leftarrow SinglePoint - 1$
    $newPopulation \leftarrow newPopulation + 1$
  end while
  for $i \leftarrow 1$ *to* $SizePopulation$ **do**
    if $Population[x_i]$ *not* $Best_1$ *or* $Best_2$ **then**
      $MutationAlgorithm(Population[x_i])$
    end if
  end for
  for $i \leftarrow 1$ *to* $SizePopulation$ **do**
    $Population[x_i] \leftarrow BalancerAlgorithm(Population[x_i])$
  end for
end while

sequentially according to the formula of PSU described earlier. If VM share at the current point is greater than zero, then it means that job size can be subtracted from the VM share. But after the subtraction if the VM share drops below zero then the job is re-added. It shows that job size is not subtracted, and job is not assigned to that VM. The unassigned jobs for which the VM share dropped below zero is added to cloudlet pool. The cloudlet pool is later used in algorithm to assign the unassigned jobs such that the workload is balanced among VMs. The subtraction of job size, when VM share does not drop below zero, shows that job is suitable for assignment to that VM and there would be no change in mapping for that particular job. The process of checking the genes of chromosome continues till the last gene and wherever the VM share goes below zero whether before subtraction or after subtraction of job size, the job is added to the cloudlet pool. This overall process ensures that no VM gets any extra job exceeding its share.

The remaining VM share of each VM is considered and according to it the VMs are sorted in descending order of their remaining share. The jobs in cloudlet pool are also sorted according to their sizes. The sub-process of balancer GA is to assign the unassigned jobs which created over utilization of VM at first place. The unassigned jobs are mapped to VMs where VM share used does not exceed the limit of share. The maximum size job in the cloudlet pool is mapped to the VM with maximum remaining VM share and then both lists of remaining VM share and cloudlet pool are sorted in descending order. The sub-process continues till there is any job left in cloudlet pool. The balancer operator of BGA is presented in algorithm 6.

3.4.2 Parameters Setting of BGA

The parameters setting of BGA are reflected in Table 3.11.

Algorithm 6 Balancer

Input: $Population[x_i]$
Output: $Population[x_i]$

procedure BALANCER
  $SizeChromosome \leftarrow size(Population[x_i])$
  for $i \leftarrow 1$ *to* $SizeChromosome$ **do**
    if $VMShare[Gene[x_i]] > 0$ **then**
      $VMShare[Gene[x_i]] \leftarrow VMShare[Gene[x_i]] - JobSize[x_i]$
      if $VMShare[Gene[x_i]] < 0$ **then**
        $VMShare[Gene[x_i]] \leftarrow VMShare[Gene[x_i]] + JobSize[x_i]$
        $list[\,] \leftarrow jobID$
      end if
    end if
  end for
  $Jobs[\,] \leftarrow sort(Jobs)$
  $VMShare \leftarrow sort(VMShare)$
  while $list[\,]$ *not empty* **do**
    while $VMShare[max] > 0$ **do**
      $VMShare[max] \leftarrow VMShare[max] - Job[list[x_i]]$
      $Population[Job[x_i]] \leftarrow VMnumber$
    end while
  end while
end procedure

TABLE 3.11: BGA Parameters

Parameter	**Type/Value**
Encoding	Discrete
Optimization	Multi Objective
Population Size	120
Stopping Criteria	Number of Iterations - 1000
Crossover Probability	Greater than 98% and 2 elites
Crossover Operation	20% Single Point 80% Multi Point
Type of Multi-point	Two Point Crossover
Mutation Rate	0.0047
Mutation Type	Random Resetting
Mutation Checked	Probability of each gene

3.5 SGA

The symbiotic GA is presented with the intention of improving the convergence speed of GA. It has been observed that GA has good potential of finding the global

optimum as described earlier in this chapter, but GA suffers slow convergence [90]. Due to this fact there should be a way to boost up the convergence process of GA. Speeding up GA may result in premature convergence and therefore the speed up process should take care that GA would not exploit too much in the search process. The global optimum should be considered in the search of solution and the convergence should improve parallel to it.

In the literature [2, 8], multiple mechanisms have been presented to improve the search speed of GA in task scheduling problem. The problem with them is that they compromise over the search of global optimum. The GA has been fused with variety of meta-heuristics to overcome any possible vulnerability. It has never been explored to merge GA with symbiosis operator. The presented technique introduces a new way in which GA is combined with the symbiotic organism search (SOS) algorithm. There are many possible ways to combine GA with other techniques and some possibilities have been revealed earlier in the thesis. One way is to keep GA as the theme approach and adding any other operation in GA which can improve the exploitation or exploration capabilities of GA. For this mutualism operator of SOS is merged with GA to improve the convergence. The working of SOS and the rationale for opting the operator of SOS are described below.

3.5.1 SOS

The SOS algorithm was proposed as a meta-heuristic in [94]. The SOS was used for task scheduling in papers [5, 11, 20, 95]. The SOS algorithm exhibits the symbiosis relationship among the organisms. The three operators of SOS are mutualism, commensalism and parasitism. The mutualism phase exhibits the symbiotic relations in which two organisms interact with each other for the mutual benefit and here both get the benefit. The Equations 3.10, 3.11 of mutualism is designed in such a way it enables the interaction of two organisms and with the reference of the best organism, both organisms are updated. The DSOS algorithm has valuable symbiotic competence to exploit by referring to the distance of any organism or particle from the best particle based on the fitness value. The mutualism phase

of DSOS updates the organism to get benefit from two organisms in which both gets either full or partial benefit.

$$x_i'(q) = \left(ceil \left(x_i + r_1(x^{best} - (\frac{x_i + x_j}{2})f1) \right) \right) mod(m+1) \tag{3.10}$$

$$x_j'(q) = \left(ceil \left(x_j + r_2(x^{best} - \left(\frac{x_i + x_j}{2} \right) f2) \right) \right) mod(m+1) \tag{3.11}$$

The r_1 and r_2 are random values generated between 0 and 1 to scale the outcome of the mutual benefit. The f1 and f2 are the random value which can be either 1 or 2. The 1 represents the partial benefit because the average of both gene is considered. On the other hand, the 2 represents the full benefit because the average value is cancelled by 2 and sum of both remains full. The ceil function is used to keep the discrete nature of the equation and modulus make sure that values may not exceed the limit. Just like in chromosome the value of gene cannot exceed the number of VMs, similarly the modulus in equations of mutualism serves the same purpose.

From the literature and experimentally it has been revealed that SOS has good exploitation capabilities. Experimentally it has been revealed that mutualism also improves the exploitation of GA which improves convergence and the exploitation of best chromosome found so far. From GA point of view the organisms can be considered as a chromosome because both have same representation. The flow of the presented SGA is presented in Figure 3.7.

The Symbiotic GA is presented in algorithm 7 and the mutualism phase of SOS begins to work after the mutation phase of GA. The fitness, selection, crossover, mutation and other parameter settings are same as presented earlier in the algorithm 1,2,3 and 4 respectively.

The Equations 3.10, 3.11 are used to compute the updated value of the chromosome. The working of mutualism is as follows. Using the Equations 3.10 and 3.11 two new chromosomes are generated with the mutual relationship of two

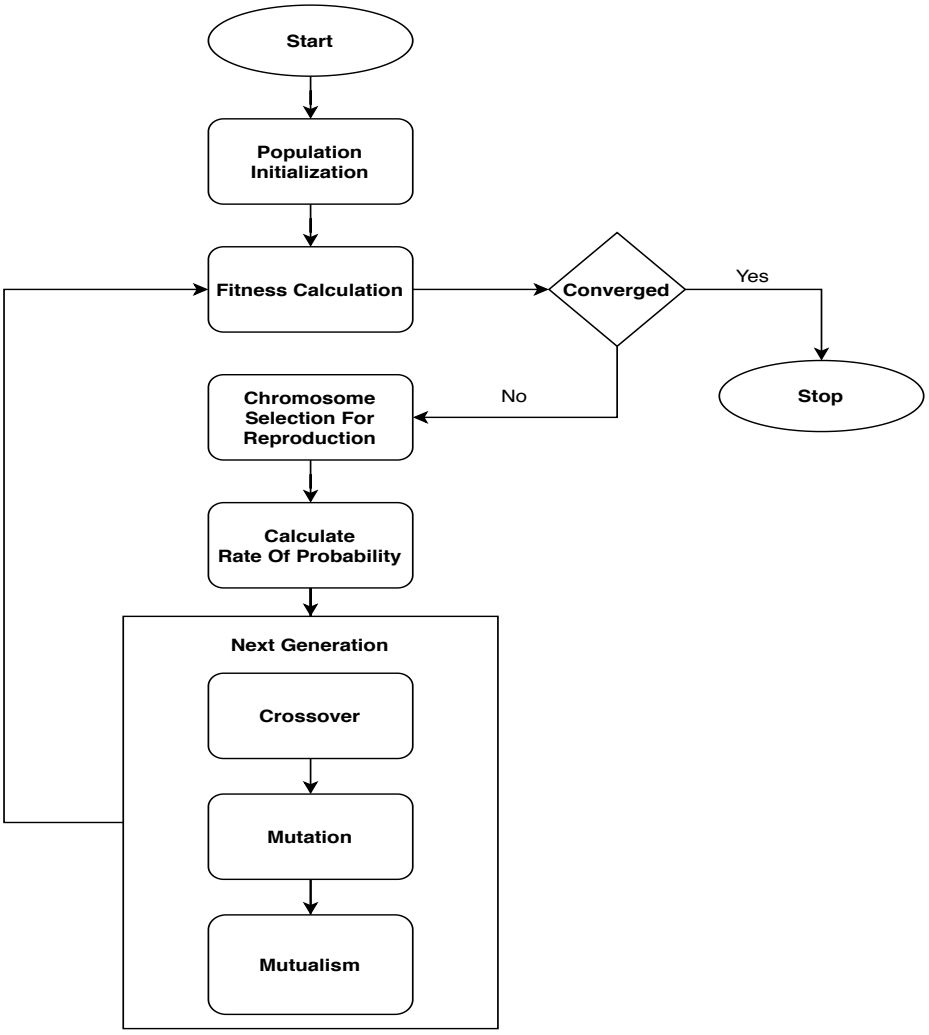


FIGURE 3.7: Symbiotic Genetic Algorithm

previous chromosomes and the best chromosome. The fitness value of both new chromosomes is evaluated and if anyone is better than the previous chromosome, it replaces previous chromosome. Otherwise newly generated chromosomes are discarded and the process of generating two new chromosomes continue for the whole population. The chromosome 'i' is selected in a sequence where 'j' is randomly selected as shown in the algorithm 7.

The mutualism operator is presented in algorithm 8. The mutualism phase is intensively required at the initial stages of the GA because population is too much diverse and far from the good solution. The mutualism phase may not be called at the later stages of algorithm if the technique is required to be run for large number of iterations.

Algorithm 7 Symbiotic GA

Input: Size of Jobs, Power of VMs
Output: Mapping of Jobs to VMs

$Population[\,] \leftarrow random(Num\_of\_VMs)$
for $i \leftarrow 0$ *to* n **do**
    $load\_balancer[\,] \leftarrow FitnessAlgorithm(Population[x_i])$
end for
$SizePopulation \leftarrow size(Population[\,])$
while GA does not converge **do**
    $newPopulation \leftarrow 0$
    $p \leftarrow 3$
    $s \leftarrow 1$
    $Parent_1 \leftarrow 1$
    $Parent_2 \leftarrow 1$
    $s \leftarrow (SizePopulation/2) - 2$
    $SinglePoint \leftarrow s \times 0.2$
    $MultiPoint \leftarrow s \times 0.8$
    while $newPopulation < s$ **do**
        while $Parent_1 == Parent_2$ **do**
            $Parent_1 \leftarrow SelectionAlgorithm(load\_balancer[\,])$
            $Parent_2 \leftarrow SelectionAlgorithm(load\_balancer[\,])$
        end while
        if $SinglePoint > 1$ **then**
            $flag \leftarrow 1$
            $Population[\,] \leftarrow CrossoverAlgorithm(Population[Parent_1], Population[Parent_2], flag, p)$
        else
            $flag \leftarrow 2$
            $Population[\,] \leftarrow CrossoverAlgorithm(Population[Parent_1], Population[Parent_2], flag, p)$
        end if
        $SinglePoint \leftarrow SinglePoint - 1$
        $newPopulation \leftarrow newPopulation + 1$
    end while
    for $i \leftarrow 1$ *to* $SizePopulation$ **do**
        if $Population[x_i]$ *not* $Best_1$ *or* $Best_2$ **then**
            $MutationAlgorithm(Population[x_i])$
        end if
    end for
    for $i \leftarrow 1$ *to* $SizePopulation$ **do**
        $j \leftarrow random(SizePopulation)$
        $MutualismAlgorithm(Population[x_i], Population[x_j])$
    end for
end while

Algorithm 8 Mutualism

Input: $Population[x_i], Population[x_j]$
Output: $Population[x_i], Population[x_j]$

procedure MUTUALISM
  $SizeChromosome \leftarrow size(Population[x_i])$
  $r1 \leftarrow random()$
  $r2 \leftarrow random()$
  $f1 \leftarrow random(2) + 1$
  $f2 \leftarrow random(2) + 1$
  for $i \leftarrow 1$ *to* $SizeChromosome$ **do**
    $A1[x_i] \leftarrow (Gene[x_i] + Gene_j[x_i])/2 \times f1$
    $B1[x_i] \leftarrow Best_1[x_i] - A1[x_i]$
    $B1[x_i] \leftarrow Gene[x_i] + r1 \times B1[x_i]$
    $B1[x_i] \leftarrow B1[x_i]\ modulus\ Num\_of\_VM$
    if $B1[x_i] < 0$ **then**
      $B1[x_i] \leftarrow B1[x_i] \times (-1)$
    end if
    $A2[x_i] \leftarrow (Gene[x_i] + Gene_j[x_i])/2 \times f2$
    $B2[x_i] \leftarrow Best_1[x_i] - A2[x_i]$
    $B2[x_i] \leftarrow Gene_j[x_i] + r2 \times B2[x_i]$
    $B2[x_i] \leftarrow B2[x_i]\ modulus\ Num\_of\_VM$
    if $B2[x_i] < 0$ **then**
      $B2[x_i] \leftarrow B2[x_i] \times (-1)$
    end if
  end for
  $MS \leftarrow CalculateFitness(B1[\,])$
  if $MS < load\_balancer[x_i]$ **then**
    for $j \leftarrow 1$ *to* $SizeChromosome$ **do**
      $Gene[x_j] \leftarrow B1[x_j]$
      $Population[x_j] \leftarrow Gene[x_j]$
    end for
  end if
  $MS \leftarrow CalculateFitness(B2[\,])$
  if $MS < load\_balancer[x_j]$ **then**
    for $j \leftarrow 1$ *to* $SizeChromosome$ **do**
      $Gene_j[x_j] \leftarrow B2[x_j]$
      $Population[x_j] \leftarrow Gene_j[x_j]$
    end for
  end if
end procedure

3.5.2 Parameters Setting of SGA

The parameters setting of SGA are reflected in Table 3.12.

3.6 Application and Analysis

The presented techniques are static schedulers that work in batch mode for independent tasks. Both techniques are different and can be used in different scenarios. This section provides the analysis of complexity and rationales for the use of both techniques.

The time complexity of non-deterministic approaches is usually not approximated but can be expressed in asymptotic notation like one calculated by Heba et al. in [86]. The complexity of BGA and SGA is an estimation of worst-case analysis depending on the operations of algorithm. There are 'n' tasks to be scheduled on $m/2$ VMs with the population size of 'm'. As the population size is almost double the number of VMs used in the experimentation therefore the VMs are represented with $m/2$. The GA based techniques are run 'k' number of times to find the optimal solution. The 'k' is the stopping criteria as well. The complexity of fitness function in the GA can be computed by adding up the worst-case running time which is $n+$

Table 3.12: SGA Parameters

Parameter	**Type/Value**
Encoding	Discrete
Optimization	Multi Objective
Population Size	120
Stopping Criteria	Number of Iterations - 500
Crossover Probability	Greater than 98% and 2 elites
Crossover Operation	20% Single Point 80% Multi Point
Type of Multi-point	Two Point Crossover
Mutation Rate	0.0047
Mutation Type	Random Resetting
Mutation Checked	Probability of each gene

$n/2+n/2+m(n+n/2)$. This is equal to n+mn and as the maximum is considered so the complexity of fitness function is O(mn). The crossover and mutation functions have complexity of O(n). The mutualism phase of SGA consumes $2(n+mn)$ time, which equates to $2n+2mn$. Overall the mutualism phase has complexity of O(mn). The balancer operation of BGA has n+n/2 complexity equal to O(n). The time complexity of BGA and SGA is same because only one operation is different. Time complexity is equal to $O(m^2n + k(m^3 + mn))$.

The heuristic based solutions move to a conclusion quickly but there are lot of solutions close to one found by the heuristic. In BGA heuristic is fused with meta-heuristic which also leads to a quick solution, but it also enables exploration of more points or solutions near the solution found by heuristic. It also helps to overcome the biasness of heuristics and to get better problem specific solution. The meta-heuristic based solutions are usually chosen because heuristics may fail with different type of datasets. Meta-heuristics are more suitable for NP hard problem where the search space is huge, and the optimization is required. Meta-heuristics can also achieve multi-objective optimization in much better way than heuristics. The BGA complies with the research question number 1 and 3.

The power of GA for task scheduling has already been discussed in Section 3.2 and it is known that pure GA is slow in convergence. Fast meta-heuristic is required therefore SGA is presented. The objective is to get better result in fewer iterations specifically when time is a concern. The original working of meta-heuristic is not disturbed with SGA and it is not biased toward any specific objective. Although fitness function plays it role like any meta-heuristic. The SGA is for task scheduling problem but not just limited to the sub-problem of improving makespan and load balancing. It can be used because it does not fail with any dataset and also does not suffer premature convergence. The SGA is for fast convergence and it is connected with the research question number 2.

3.7 Performance Model and Metrics

The presented technique BGA improves the makespan and resource utilization through load balancing, where the SGA technique improves the convergence speed of GA. The SGA also considers the improvement in the performance metrics of makespan and resource utilization. Although the main focus is to improve convergence such that the makespan and resource utilization are not so compromised. The performance of BGA is evaluated on makespan and resource utilization whereas convergence is focused in SGA as it coincides with the research questions. This section provides the detail of performance metrics used for the evaluation of both techniques. The formulas for the calculation of metrics are also provided and some are already described so they are referred where necessary. The presented techniques and the corresponding measures of evaluation are presented in the Table 3.13.

TABLE 3.13: Presented Techniques and Performance Measures

Techniques	**Performance Measures**
BGA	Makespan and ARUR
SGA	Curve for Makespan and ARUR

3.7.1 Makespan

The makespan is the finishing time of batch of jobs. The formula for the calculation is provided as follows [5, 7, 8, 10, 15, 16, 20, 21, 24, 34, 37, 52, 57, 59, 65].

$$CT_{VM_j} = \frac{\sum_{i=1}^{n}(jobsize_i \times Map[i,j])}{VM_j mips} \; \forall \; i \in 1,2,3,\ldots,m \;\&\; j \in 1,2,3,\ldots,n \tag{3.12}$$

$$makespan = \max\left(CT_{VMj}\right) \; \forall j \in 1,2,3,\ldots,n \tag{3.13}$$

3.7.2 ARUR

The ARUR is average resource utilization ratio and the formula is provided as follows [7, 13, 35, 44–51].

$$ARUR = \frac{(\sum_{j=1}^{n} CT_{VM_j})/n}{makespan} \tag{3.14}$$

The value of ARUR is 1 when average finishing time of all VMs is equal to the maximum finishing time [7]. It shows that all VMs have taken equal amount of time and this is the objective of good load balancing. Load balancing does not always ensure that makespan would be minimum. The value of ARUR can be good for different possible mappings, having varying makespan values. To achieve good makespan considering the optimal load balance, both objectives are considered parallel as already discussed in the Section 3.3.3.

3.7.3 Convergence

Convergence is a measure to show the value of fitness function over the number of iterations of algorithms. The convergence curve in meta-heuristic is revealed in [2, 5, 8].

- Based on makespan
- Based on ARUR

3.8 Evaluation Benchmarks

There are many heuristics, meta-heuristics and hybrid techniques proposed for efficient task scheduling. Some are reported in well reputed journals and conferences as described in the literature review. This section provides the detail of baseline papers for the comparison of results of the presented techniques.

The reason for the selection of those papers is also highlighted. The main concerns in the selection of papers are to choose the recent publications which are most relevant to this research. In the recent years many GA based techniques [2, 8, 14, 17, 18, 23, 24, 44, 52, 54, 55] have been proposed so for the comparison of SGA whose main focus is on the convergence the evolutionary and metaheuristic approaches are selected. These approaches are ETA-GA [52] and DSOS [5]. BGA has a heuristic balancer operation therefore both heuristic and metaheuristic techniques are used in the comparison. These approaches are ETA-GA [52], MGGS [14], DSOS [5] and RALBA [7]. Their results are reported in the publications and mostly dataset of the papers is not available. But according to the set claims in the research questions, benchmark techniques are opted. The details of dataset for the comparison of performance and rationale are described in the next section.

The ETA-GA [52] is a pure GA for task scheduling, and it is compared with both BGA and SGA. When compared with SGA the convergence is considered in achieving the good fitness value. Similarly, when compared with BGA both makespan and resource utilization are compared. The ETA-GA is published in "cluster computing" journal in 2019 with objective of improving makespan. The results of ETA-GA are reported but the dataset is not available. The ETA-GA is not open source therefore for the comparison the technique is implemented following the instructions available in the research paper. The parameter settings of ETA-GA is exactly same as referred in the literature and the iterations of ETA-GA and presented techniques are fixed for comparison on equal grounds.

The MGGS [14] is a binary GA having a heuristic operation of load balancing and it fits with the objectives of BGA for comparison of resource utilization and makespan. The MGGS is published in "neural computing and applications" journal in 2019 and has objective of improving makespan and load balancing, where load balacing is not part of fitness function. The MGGS is compared with BGA. The iterations of MGGS are computationally expensive and the iterations can take lot of time in MGGS in the worst case. The presented BGA is a GA based technique having heuristic just like MGGS but the operations of BGA are light weight. The results of MGGS are reported but it is evaluated on a very small dataset and

some instances of the dataset are available. It is not right choice to evaluate the presented technique on the same small dataset. Therefore, the MGGS is implemented according to the details mentioned in the research publication. Using the available instances of the synthetic dataset presented in the paper the results of the implemented MGGS are generated which are same as mentioned in the paper. It shows that the implementation is accurate and can be used for the sake of comparison. The MGGS is not compared with SGA because the agenda of MGGS complies with BGA but purpose of SGA is improvement in convergence. Whenever heuristic is embedded with meta-heuristic, the meta-heuristic is guided with the heuristic and rushes toward heuristic solution. Therefore, the convergence is by default fast, but the comparison of convergence is not for such type of techniques. Instead the convergence analysis is for meta-heuristics and the advantage of pure meta-heuristic is discussed in Section 3.6. Having said that one another reason for not comparing MGGS with SGA is that the MGGS takes too much running time specially when large sized batches are considered. MGGS converges to a solution in approximately 200 iterations for large batches and afterward it does not improve. The 200 iterations of MGGS takes time almost equal to 4000 iterations of SGA. Whereas the SGA keeps improving after 4000 iterations and so on. Hence, the SGA cannot be compared with MGGS on equal grounds considering the convergence analysis.

The DSOS [5] is a meta-heuristic approach for task scheduling and it is published in "future generation computer systems" journal in 2016. Fitness function in DSOS is based on makespan whereas fast convergence is also exhibited as an achievement. The performance of DSOS is evaluated on a synthetic dataset which is not available in the paper, but the same dataset is used by the authors in another publication. In another publication the dataset is available. To evaluate the performance of DSOS the technique is implemented. For the proof of correct implementation, the results using the above-mentioned synthetic dataset are generated and they are comparable with the results presented in the paper. The DSOS is compared with both SGA and BGA.

The RALBA [7] is a heuristic approach, published in "cluster computing" journal in 2018. RALBA shows improvement over other popular heuristics and therefore the comparison of the presented technique is required to get idea how the presented technique is performing. RALBA is compared with BGA only because it cannot be compared for convergence as it is a heuristic approach. RALBA is an open source technique and the dataset used in the RALBA is a large realistic dataset which is also published. The same dataset is used for performance evaluation of BGA and the parameter settings is consistent for comparison of makespan and resource utilization. There is no intention of comparing the presented techniques with each other as they have different objectives. The Table 3.14 shows the presented technique and the benchmark techniques for comparison whereas the details of each paper is already described above.

TABLE 3.14: Bechmark Techniques

Presented Techniques	**Benchmark Techniques**		
	Heuristic	**Meta-Heuristic**	**Hybrid**
BGA	RALBA	DSOS	MGGS
		ETA-GA	
SGA		DSOS	
		ETA-GA	

3.9 Dataset

The dataset for independent task [7, 96] scheduling consists of tasks and VMs with their sizes and power specifications respectively. The main portion of the dataset is the detail of tasks to be mapped and different VM specifications are chosen for experimentation by the researchers. There are many techniques in the literature which have been tested on sample datasets having very small sizes of tasks and few numbers of instances [14, 17, 23, 29, 38, 40, 57, 72, 75, 76, 85]. Such small sizes and fewer instances of tasks may not be the right choice to evaluate the performance of scheduler because the cloud-based systems usually have large number of computationally intensive tasks [94]. Normally there are two types of datasets that are used in independent task scheduling.

1. Synthetic Test Dataset
2. Realistic Test Dataset

The presented techniques are evaluated on both synthetic and realistic datasets to avoid dataset dependent findings. The details of the datasets are provided below.

3.9.1 Synthetic Dataset

The dataset is regarded as synthetic if it is not gathered from the original event [87]. By original event in case of tasks, it is meant that data is based on the real traces of any cloud system. So, synthetic dataset is made analyzing the real traces but not actually derived from the real traces. Hence, the synthetic dataset is not specific to any cloud system and it is more of generic representation than specific to any cloud. The synthetic dataset used for evaluation of the presented techniques have been used in the following literatures [5, 11, 95]. The dataset consists of four classes and these are:

1. Left Skewed
2. Right Skewed
3. Normal or bell shaped
4. Uniform

Each category of the defined dataset has batch of jobs and there are ten different batch sizes. The batches are of 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1000 jobs. The job sizes vary in each batch size the Table 3.15 shows a sample having few instances of left skewed dataset where batch size is 1000. Left skewed dataset has more large size tasks than small tasks whereas in right skewed it is opposite to left skewed. Normal distribution has some small and large size tasks but mostly tasks are of medium size. The normal distribution is bell shaped. The uniform distribution represents that small, medium and large tasks are mostly equal in number.

TABLE 3.15: Sample of Synthetic Dataset

Job no.	Job size (mi)
1.	34304
2.	37849
3.	36347
4.	30933
5.	34976
6.	38501
7.	37059
8.	36504
9.	29606
10.	34944

3.9.2 Realistic Dataset

The dataset is called to be realistic dataset if it is based on the real events [87, 89]. The traces of cloud normally have the execution time and realistic dataset should have sizes of tasks obtained from the execution timings.

For the evaluation of the presented techniques the realistic dataset having name Google Cloud Jobs (GoCJ) [89] is used. The same dataset has been used in the following publications [7, 13]. The realistic dataset has tasks with their sizes, and they are based on the traces of Google cluster and M45 supercomputing cluster. To generate the workload the Monte-Carlo simulation has been deployed [89]. The batch sizes available in the dataset are 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, and 1000.

Some of the initial instances of batch size 1000 are presented in the Table 3.16.

Specifications

Normally the power of VMs are chosen randomly for experimentation. The unit for the power of VMs is million instructions per second (MIPS). For the evaluation of the presented techniques the parameter list of VMs is presented in the Table 3.17 and same has been used in [7].

TABLE 3.16: Sample of Realistic Dataset

Job no.	**Job size (mi)**
1.	83000
2.	95000
3.	91000
4.	81000
5.	27500
6.	49000
7.	103000
8.	95000
9.	47000
10.	71000

TABLE 3.17: VM Specifications

VM no.	**Processing Power (MIPS)**	**VM no.**	**Processing Power (MIPS)**	**VM no.**	**Processing Power (MIPS)**
1.	100	21.	1000	41.	1750
2.	100	22.	1000	42.	1750
3.	100	23.	1000	43.	1750
4.	100	24.	1000	44.	1750
5.	100	25.	1000	45.	4000
6.	100	26.	1000	46.	4000
7.	100	27.	1250	47.	4000
8.	500	28.	1250	48.	4000
9.	500	29.	1250	49.	4000
10.	500	30.	1250	50.	4000
11.	500	31.	1250		
12.	500	32.	1250		
13.	500	33.	1500		
14.	500	34.	1500		
15.	750	35.	1500		
16.	750	36.	1500		
17.	750	37.	1500		
18.	750	38.	1500		
19.	750	39.	1750		
20.	750	40.	1750		

3.10 Experimental Setup

The experimentations are performed with the configurations presented in the section below.

3.10.1 System Configurations

The machine used for the experimentation possesses the 1.7 GHz clock speed on Intel Core 3-4010U Dual Core CPU and the main memory of 4 GBs. The simulator used for deploying the cloud system is CloudSim 3.0.3 and the simulation environment is presented in the Table 3.18.

TABLE 3.18: System Configurations

Datacenter	1
Total Host Machines	30
Cloud Host Machines	4 Dual-core and 26 Quad-core machines each of 4000 MIPS
Memory of Host Machines	Each of 16384 MBs
Host Bandwidth	10000 Megabits/s
Host Storage	1000000 MB
Count of VMs	50 VMs
Processors in VMs	1
Operating System	Linux
System Architecture	x86
Virtual Machine Monitor (VMM)	Xen
Image Size	1000 MB
VM RAM	512 MBs
VM Bandwidth	1000 Megabits/s
Cloudlet Scheduler	Space Shared

The benchmark techniques used for the evaluation are already described. They have their own parameter settings, but few settings are made consistent for investigating on equal grounds. The parameter settings for comparing BGA and other techniques are 1000 number of iterations and 120 number of chromosomes. SGA is compared with meta-heuristics and number of chromosomes are set to 120 and the iterations are 500. Each meta-heuristic technique is run three times as they are stochastic. The average of three run is utilized in performance evaluation.

Chapter 4

Experimentation and Evaluation

This chapter provides the experimentation performed for the evaluation of presented techniques. All the benchmark techniques as described earlier are evaluated on two datasets and results are compared with BGA and SGA. The experimental setup and environment are described earlier. First the experimentation conducted to set the mutation rate of BGA, and SGA are demonstrated. Later the performance of BGA and SGA tested on realistic and synthetic datasets is presented with the necessary discussion. The makespan and resource utilization of both techniques are presented in experimentations. As there are different batch sizes in both datasets therefore multiple graphs are generated for different batch sizes. The average of all batch sizes for the performance metrics is also separately provides in graphs. The improvement or decline in findings of evaluated approaches is also highlighted.

4.1 Mutation Rate

The BGA and SGA are based on GA and the one of the important aspects in fine tuning of genetic based technique is the setting of mutation rate. Different mutation rates are tested on GoCJ workload, batch of 1000 for 500 iterations. The findings are presented in Figure 4.1 and 4.2.

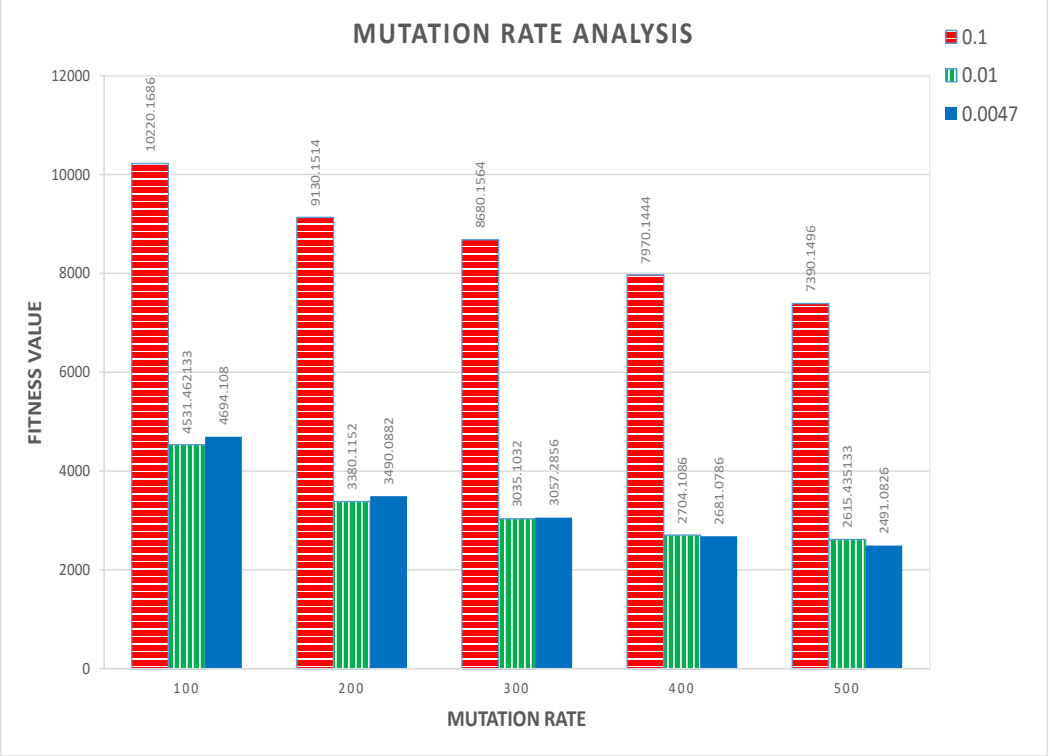


FIGURE 4.1: Mutation Rate Analysis - A

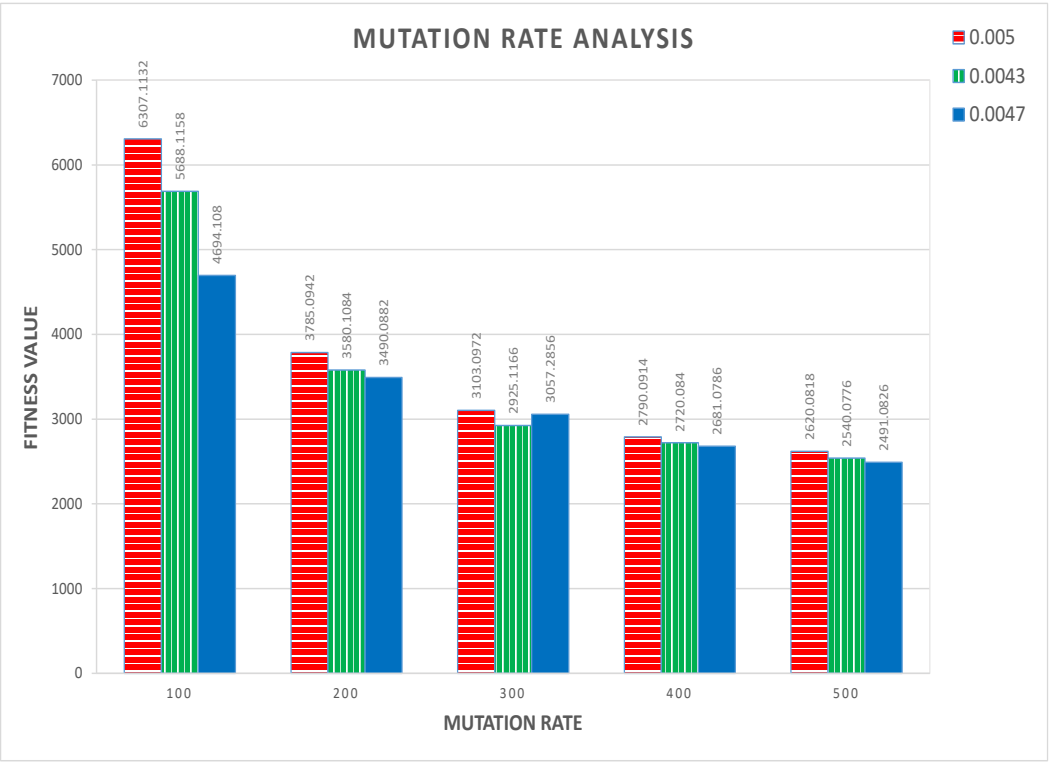


FIGURE 4.2: Mutation Rate Analysis - B

To set the mutation rate for the presented techniques the population of GA is analyzed. Different experimentations are conducted to analyze the behavior of GA with varying mutation rates in the problem of task scheduling. It has been observed during the experimentation that making mutation rate adaptive or variable cannot have significant impact and over the iteration the fitness results in same value. Whereas setting static mutation can contribute good enough in the problem of task scheduling. As long as the population is diverse there is no need to change the mutation rate but there should be a balance in the diversity of population as discussed in the Section 3.3.6. The fitness value is a key factor which shows that either fitness is improving or not. For different set of mutation rates the GA demonstrates continuous improvement but it is required that the fitness value should improve quickly and continuously. Same mutation rate set experimentally is applied in both BGA and SGA. Initially the mutation rate was set to 0.1 and then it was increased to 0.2. The performance degraded and therefore the rate was later decreased to 0.05 and 0.01. The GA showed improvement with the decrease in mutation rate as shown in the Figure 4.2. At reaching mutation rate of 0.005 the performance becomes more stable and finally at 0.0047 the GA exhibits optimal performance for 500 iterations of GA. Later in all experimentations the mutation rate is set to 0.0047 in both BGA and SGA.

4.2 BGA

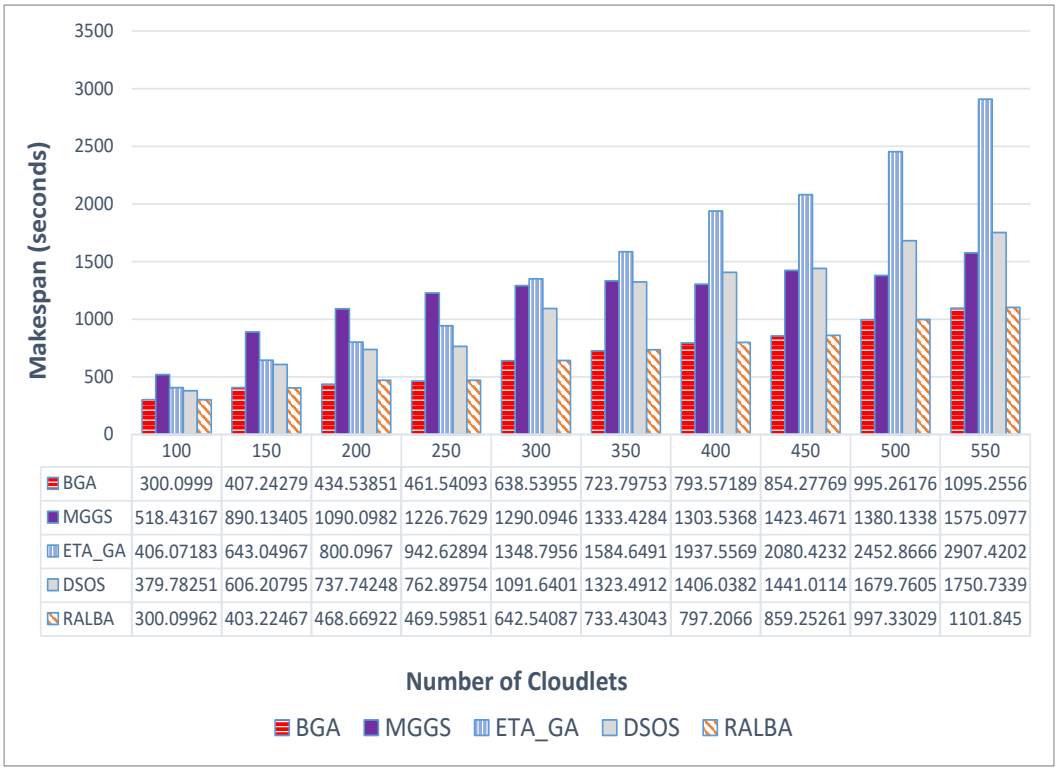
The performance of BGA compared to other benchmark techniques is exhibited based on the makespan and ARUR. Each technique is evaluated on average of three experiments as the meta-heuristics are stochastic in nature. The MGGS, ETA-GA, DSOS and RALBA are compared with BGA for makespan and ARUR.

4.2.1 Makespan

The makespan of BGA is analyzed on GoCJ and Synthetic datasets, whereas in synthetic dataset all categorizes namely left, right, normal and uniform workload

are utilized in experimentations.

The Figure 4.3 shows the makespan of BGA using GoCJ dataset on batch sizes of 100 to 550 with the difference of 50 in batch sizes. BGA has achieved better makespan in most of the cases. At batch of 100 and 150 the BGA and RALBA are almost same, while in all other batch sizes the BGA has achieved better makespan as compared to MGGS, ETA-GA, DSOS and RALBA. The MGGS shows worst performance for small batch sizes as shown in Figure 4.3. The ETA-GA does not show good makespan on increasing the batch size and its performance declined on batch of 300 and onwards.

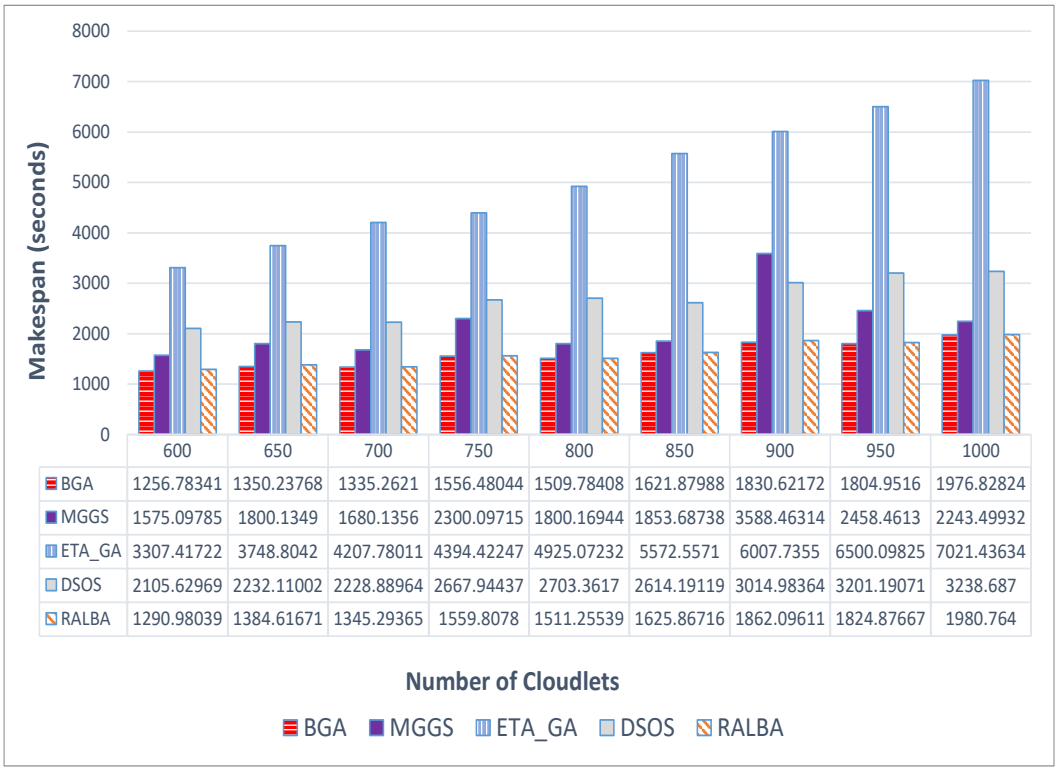


	100	150	200	250	300	350	400	450	500	550
BGA	300.0999	407.24279	434.53851	461.54093	638.53955	723.79753	793.57189	854.27769	995.26176	1095.2556
MGGS	518.43167	890.13405	1090.0982	1226.7629	1290.0946	1333.4284	1303.5368	1423.4671	1380.1338	1575.0977
ETA_GA	406.07183	643.04967	800.0967	942.62894	1348.7956	1584.6491	1937.5569	2080.4232	2452.8666	2907.4202
DSOS	379.78251	606.20795	737.74248	762.89754	1091.6401	1323.4912	1406.0382	1441.0114	1679.7605	1750.7339
RALBA	300.09962	403.22467	468.66922	469.59851	642.54087	733.43043	797.2066	859.25261	997.33029	1101.845

FIGURE 4.3: Makespan on GoCJ Dataset 100 to 550

The makespan of BGA using GoCJ dataset on batch sizes 600 to 1000 are exhibited in Figure 4.4. The ETA-GA declined further on large batch sizes. The BGA has attained better makespan than all benchmark techniques. The MGGS takes lot of time on large batch sizes and also improves but the makespan of BGA is still better than MGGS. Figure 4.5 shows the average of all batch sizes to demonstrate the overall behavior in terms of makespan on GoCJ dataset. BGA has achieved

33.1, 65.5, 40.4 and 1% average improvement in makespan than MGGS, ETA-GA, DSOS and RALBA respectively.



	600	650	700	750	800	850	900	950	1000
BGA	1256.78341	1350.23768	1335.2621	1556.48044	1509.78408	1621.87988	1830.62172	1804.9516	1976.82824
MGGS	1575.09785	1800.1349	1680.1356	2300.09715	1800.16944	1853.68738	3588.46314	2458.4613	2243.49932
ETA_GA	3307.41722	3748.8042	4207.78011	4394.42247	4925.07232	5572.5571	6007.7355	6500.09825	7021.43634
DSOS	2105.62969	2232.11002	2228.88964	2667.94437	2703.3617	2614.19119	3014.98364	3201.19071	3238.687
RALBA	1290.98039	1384.61671	1345.29365	1559.8078	1511.25539	1625.86716	1862.09611	1824.87667	1980.764

FIGURE 4.4: Makespan on GoCJ Dataset 600 to 1000

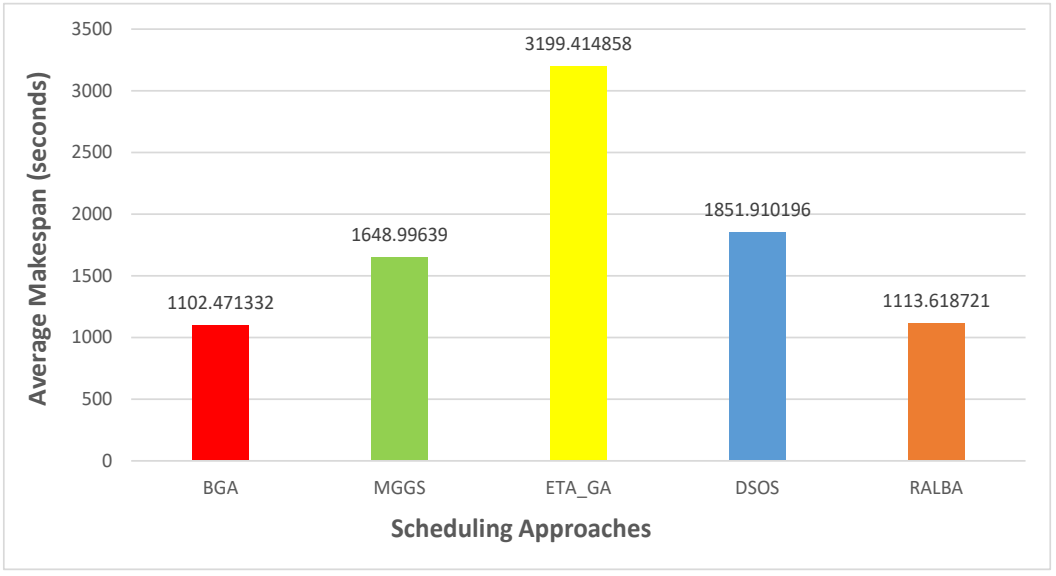
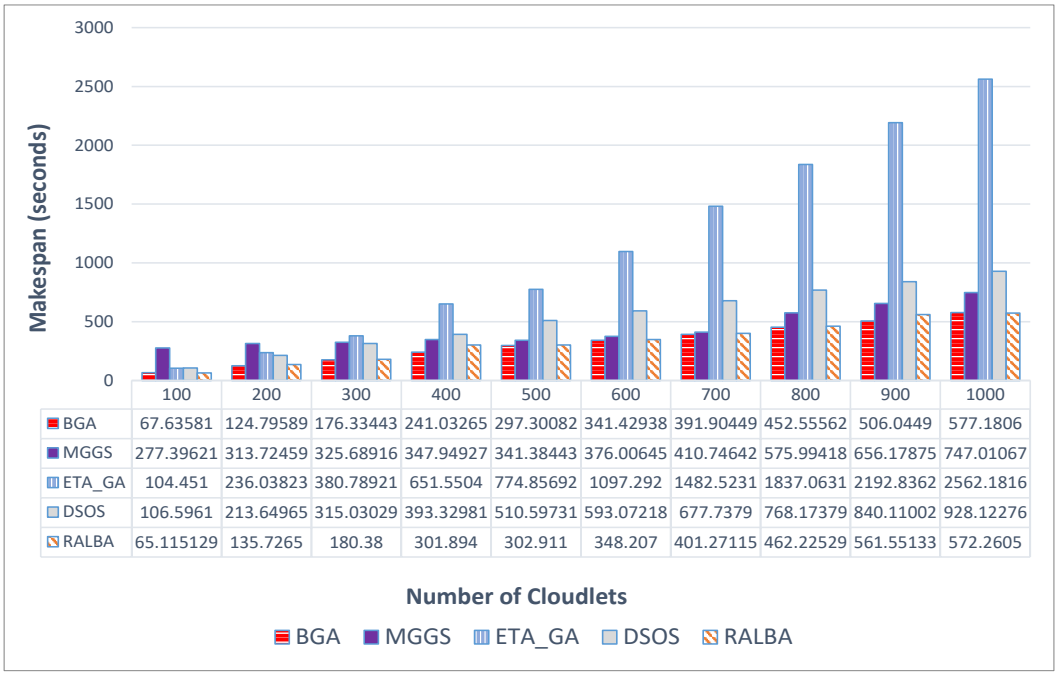


FIGURE 4.5: Average Makespan on GoCJ Dataset

The makespan is analyzed on all categories of synthetic dataset for batch sizes of 100 to 1000 with the difference of 100 in batch sizes. Figure 4.6 shows the analysis

of makespan on left skewed dataset and most of the times the makespan of BGA is better than other techniques. RALBA shows better makespan at batch size 100 and 1000, other than that the makespan of BGA is better than RALBA. The Figure 4.7 exhibits the average of all batch sizes to show the trend of improvement in BGA. On average the BGA has achieved 27.3, 71.9, 40.5, and 4.6% better makespan than MGGS, ETA-GA, DSOS and RALBA respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	67.63581	124.79589	176.33443	241.03265	297.30082	341.42938	391.90449	452.55562	506.0449	577.1806
MGGS	277.39621	313.72459	325.68916	347.94927	341.38443	376.00645	410.74642	575.99418	656.17875	747.01067
ETA_GA	104.451	236.03823	380.78921	651.5504	774.85692	1097.292	1482.5231	1837.0631	2192.8362	2562.1816
DSOS	106.5961	213.64965	315.03029	393.32981	510.59731	593.07218	677.7379	768.17379	840.11002	928.12276
RALBA	65.115129	135.7265	180.38	301.894	302.911	348.207	401.27115	462.22529	561.55133	572.2605

FIGURE 4.6: Makespan on Left Skewed Dataset

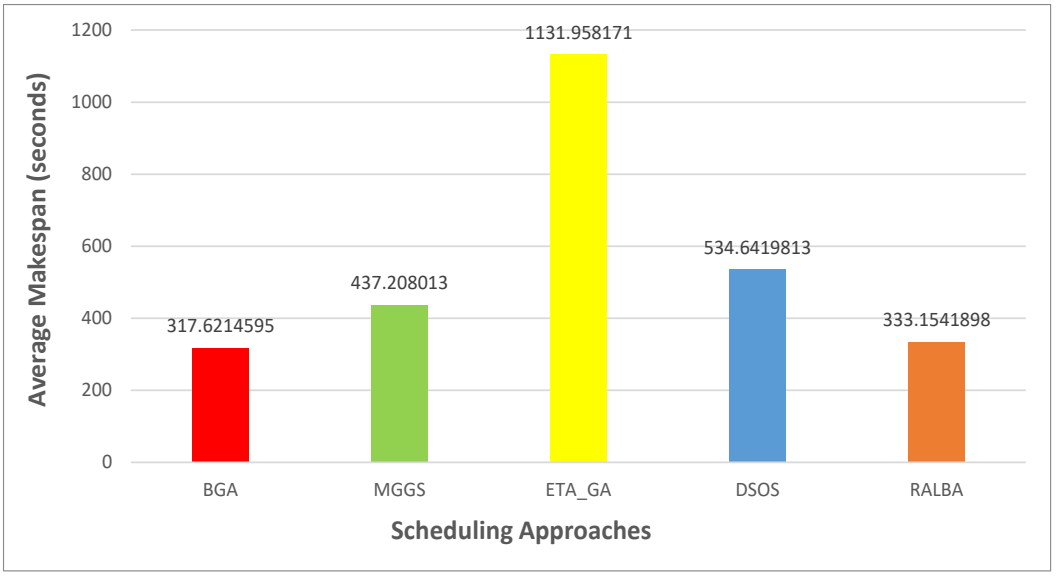
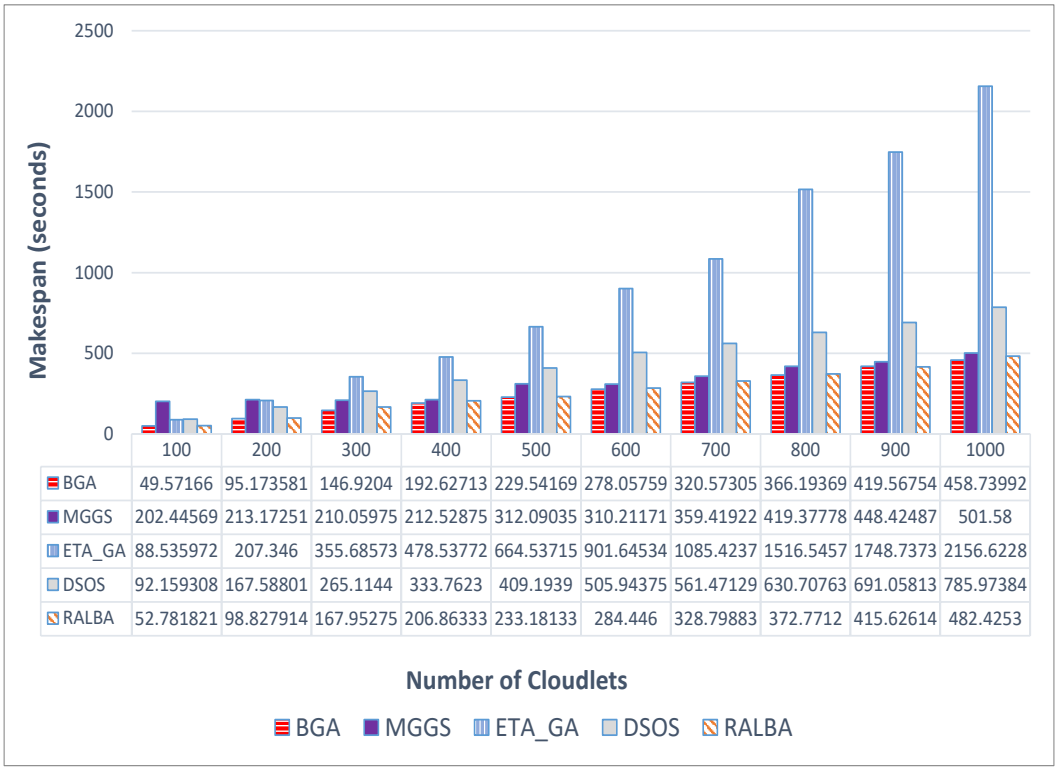


FIGURE 4.7: Average Makespan on Left Skewed Dataset

The Figure 4.8 shows the makespan analysis on right skewed dataset. It is evident that in most of the batch sizes the BGA showed better makespan. Figure 4.9 shows that BGA has attained 19.8, 72.2, 42.4, and 3.2% improvement in makespan than MGGS, ETA-GA, DSOS and RALBA respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	49.57166	95.173581	146.9204	192.62713	229.54169	278.05759	320.57305	366.19369	419.56754	458.73992
MGGS	202.44569	213.17251	210.05975	212.52875	312.09035	310.21171	359.41922	419.37778	448.42487	501.58
ETA_GA	88.535972	207.346	355.68573	478.53772	664.53715	901.64534	1085.4237	1516.5457	1748.7373	2156.6228
DSOS	92.159308	167.58801	265.1144	333.7623	409.1939	505.94375	561.47129	630.70763	691.05813	785.97384
RALBA	52.781821	98.827914	167.95275	206.86333	233.18133	284.446	328.79883	372.7712	415.62614	482.4253

FIGURE 4.8: Makespan on Right Skewed Dataset

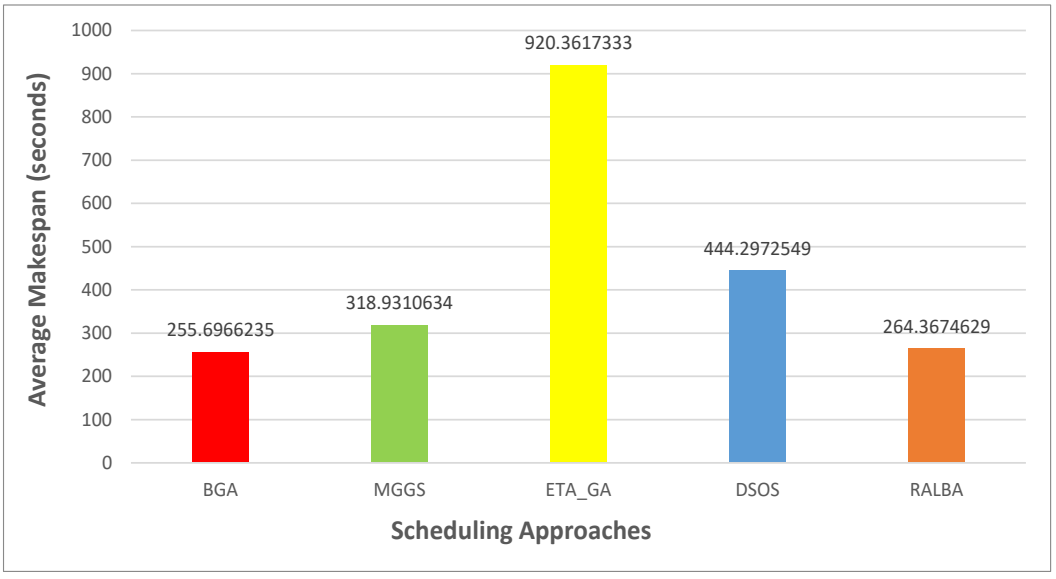
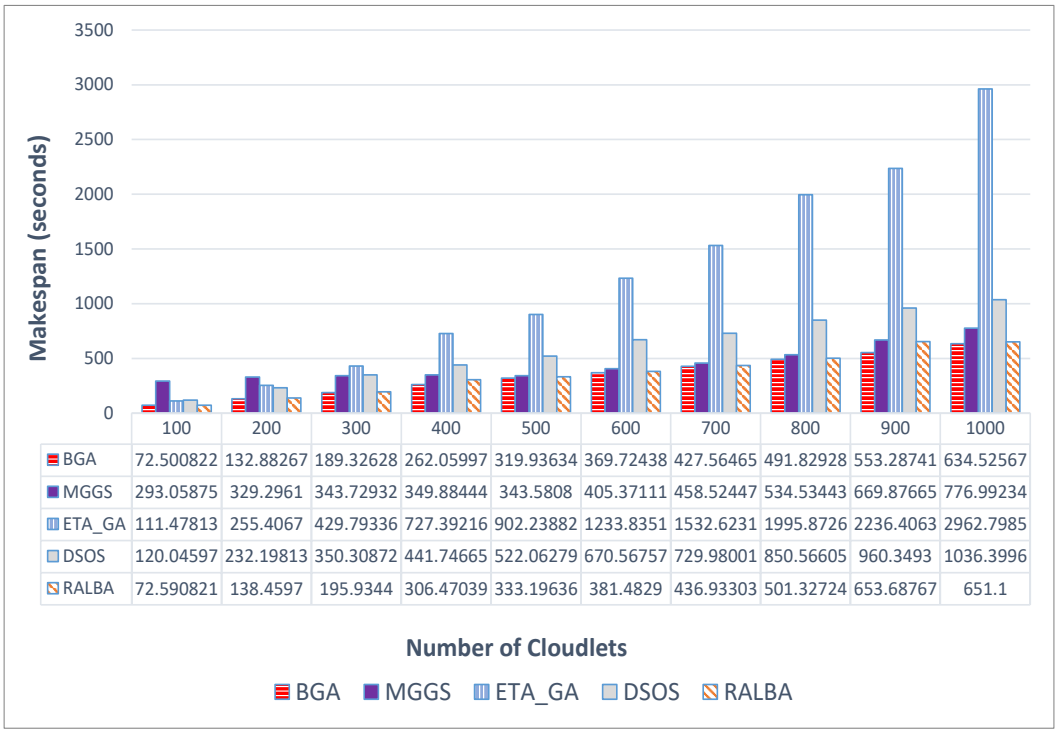


FIGURE 4.9: Average Makespan on Right Skewed Dataset

The Figure 4.10 shows the makespan analysis on normal dataset and Figure 4.11 shows average percentage of improvement of BGA against other techniques. BGA has achieved better makespan on all batch sizes of normal dataset. The improvement of BGA is 23.3, 72.1, 41.6, and 5.9% against MGGS, ETA-GA, DSOS and RALBA respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	72.500822	132.88267	189.32628	262.05997	319.93634	369.72438	427.56465	491.82928	553.28741	634.52567
MGGS	293.05875	329.2961	343.72932	349.88444	343.5808	405.37111	458.52447	534.53443	669.87665	776.99234
ETA_GA	111.47813	255.4067	429.79336	727.39216	902.23882	1233.8351	1532.6231	1995.8726	2236.4063	2962.7985
DSOS	120.04597	232.19813	350.30872	441.74665	522.06279	670.56757	729.98001	850.56605	960.3493	1036.3996
RALBA	72.590821	138.4597	195.9344	306.47039	333.19636	381.4829	436.93303	501.32724	653.68767	651.1

FIGURE 4.10: Makespan on Normal Dataset

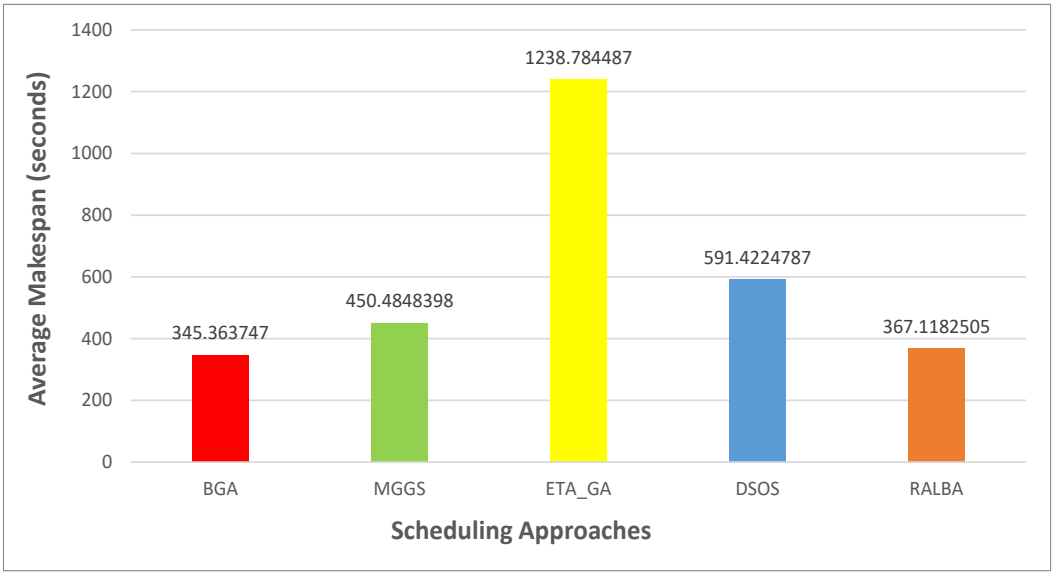
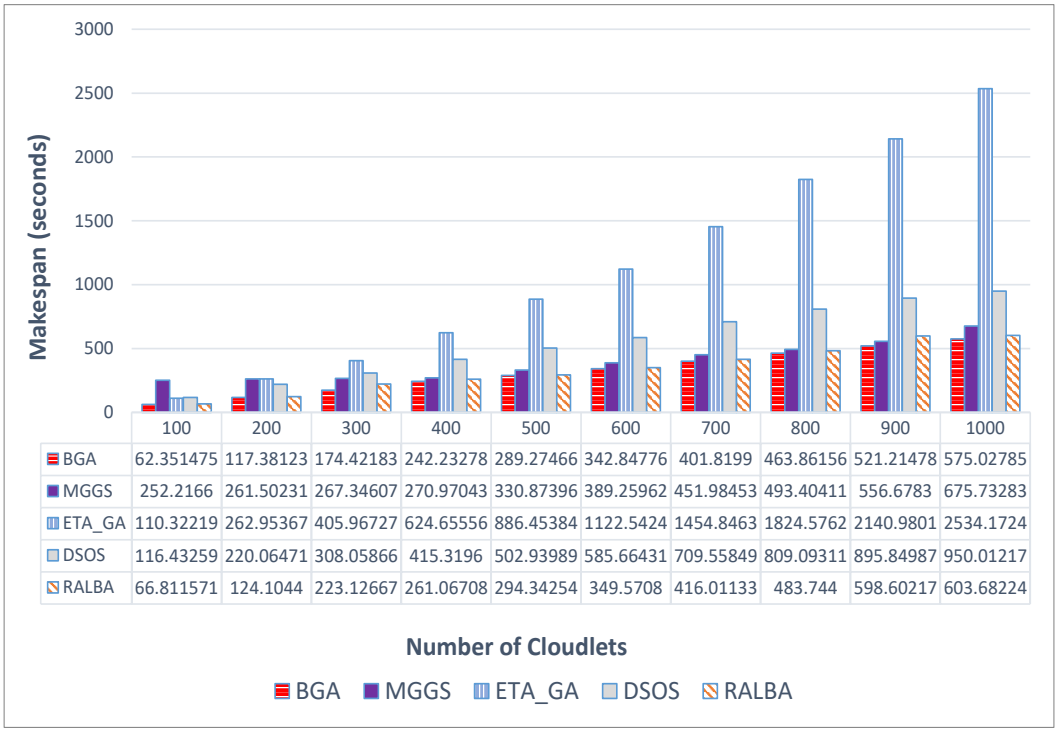


FIGURE 4.11: Average Makespan on Normal Dataset

The Figure 4.12 and Figure 4.13 demonstrate the behavior of BGA and other techniques on uniform dataset. BGA shows improvement in all batch sizes. The percentage of improvement in makespan against MGGS, ETA-GA, DSOS and RALBA are 19.2, 71.9, 42.1, and 6.7% respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	62.351475	117.38123	174.42183	242.23278	289.27466	342.84776	401.8199	463.86156	521.21478	575.02785
MGGS	252.2166	261.50231	267.34607	270.97043	330.87396	389.25962	451.98453	493.40411	556.6783	675.73283
ETA_GA	110.32219	262.95367	405.96727	624.65556	886.45384	1122.5424	1454.8463	1824.5762	2140.9801	2534.1724
DSOS	116.43259	220.06471	308.05866	415.3196	502.93989	585.66431	709.55849	809.09311	895.84987	950.01217
RALBA	66.811571	124.1044	223.12667	261.06708	294.34254	349.5708	416.01133	483.744	598.60217	603.68224

FIGURE 4.12: Makespan on Uniform Dataset

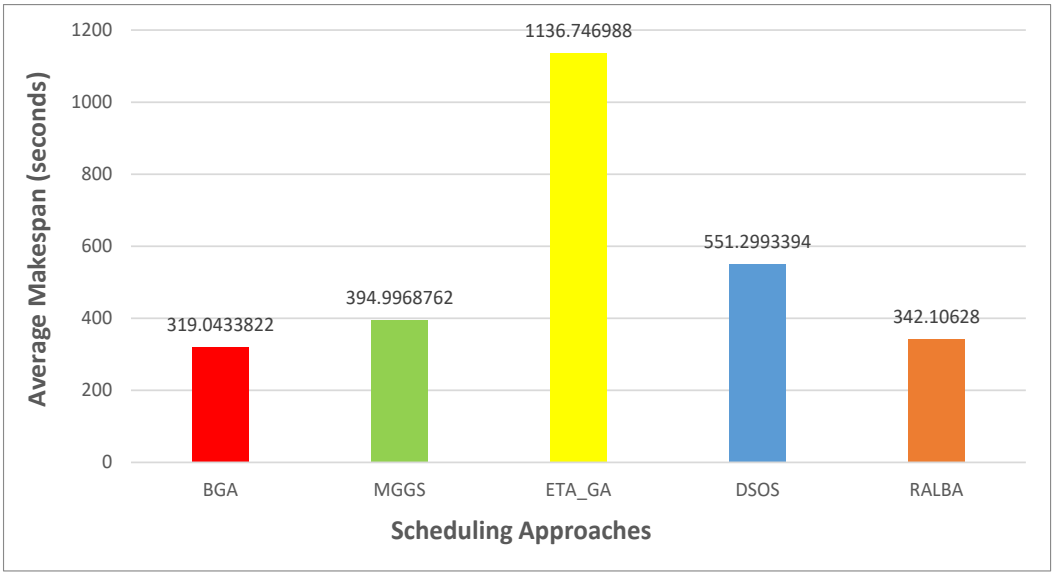
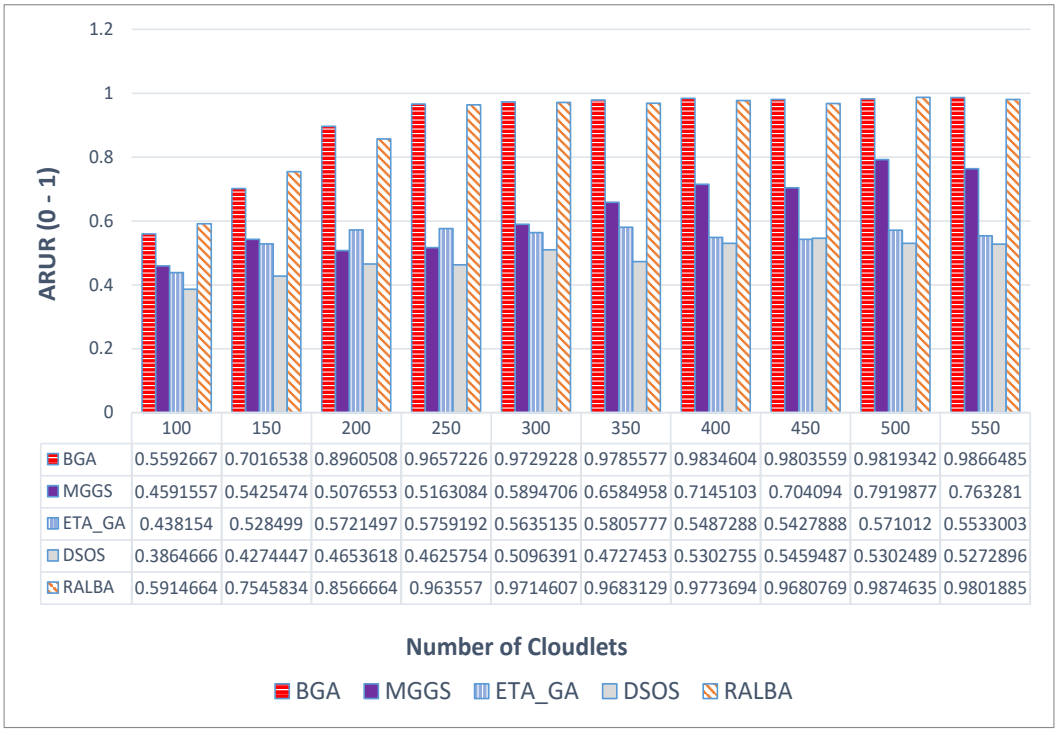


FIGURE 4.13: Average Makespan on Uniform Dataset

It is evident that BGA outperforms all other techniques on two different datasets in terms of makespan. The RALBA is a close competitor of BGA. MGGS and DSOS are good at large batch sizes whereas ETA-GA does not perform well in 1000 iterations.

4.2.2 ARUR

The ARUR is a measure to compute the resource utilization. The performance of BGA is tested on GoCJ and then on synthetic dataset. Figure 4.14 shows the ARUR of BGA and other techniques on GoCJ dataset for batch size 100 to 550 with the difference of 50 in batch sizes. RALBA shows good ARUR on batch size 100 and 200, whereas BGA outperforms on large batch sizes. Figure 4.15 shows the analysis of ARUR on GoCJ dataset with batch sizes between 600 and 1000. Overall BGA exhibits 30, 80.4, 83.1, and 0.5% improvement in ARUR against MGGS, ETA-GA, DSOS and RALBA respectively as shown in Figure 4.16. For large batch sizes the BGA has good load balancing on GoCJ dataset.



	100	150	200	250	300	350	400	450	500	550
BGA	0.5592667	0.7016538	0.8960508	0.9657226	0.9729228	0.9785577	0.9834604	0.9803559	0.9819342	0.9866485
MGGS	0.4591557	0.5425474	0.5076553	0.5163084	0.5894706	0.6584958	0.7145103	0.704094	0.7919877	0.763281
ETA_GA	0.438154	0.528499	0.5721497	0.5759192	0.5635135	0.5805777	0.5487288	0.5427888	0.571012	0.5533003
DSOS	0.3864666	0.4274447	0.4653618	0.4625754	0.5096391	0.4727453	0.5302755	0.5459487	0.5302489	0.5272896
RALBA	0.5914664	0.7545834	0.8566664	0.963557	0.9714607	0.9683129	0.9773694	0.9680769	0.9874635	0.9801885

FIGURE 4.14: ARUR on GoCJ Dataset 100 to 550

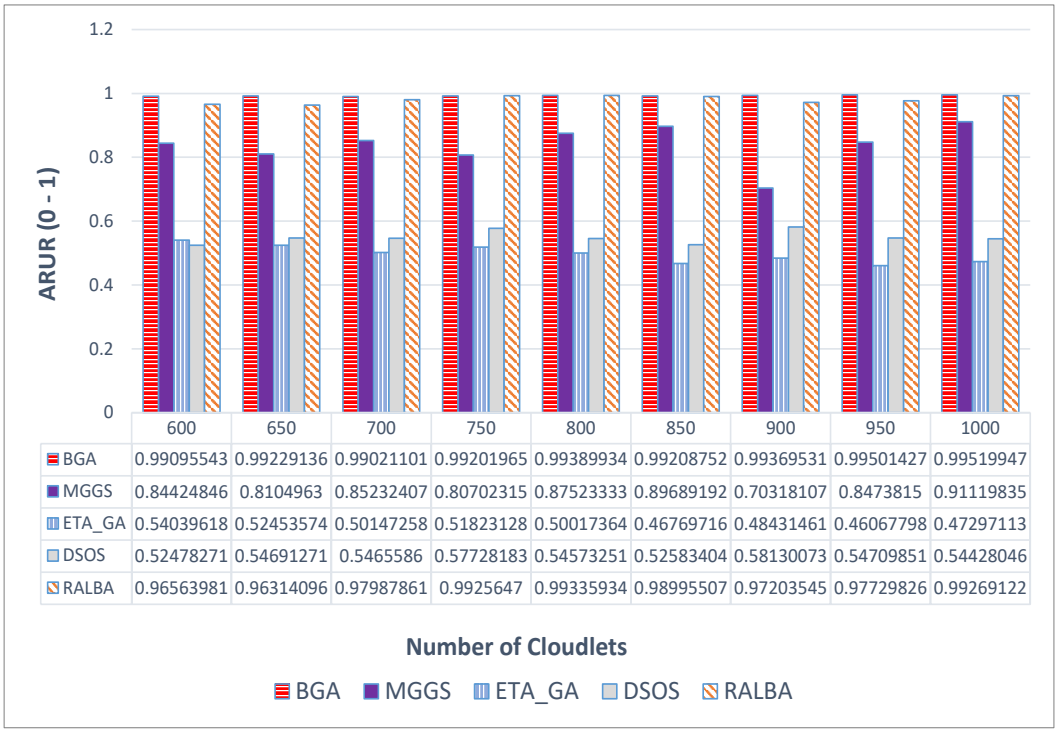


FIGURE 4.15: ARUR on GoCJ Dataset 600 to 1000

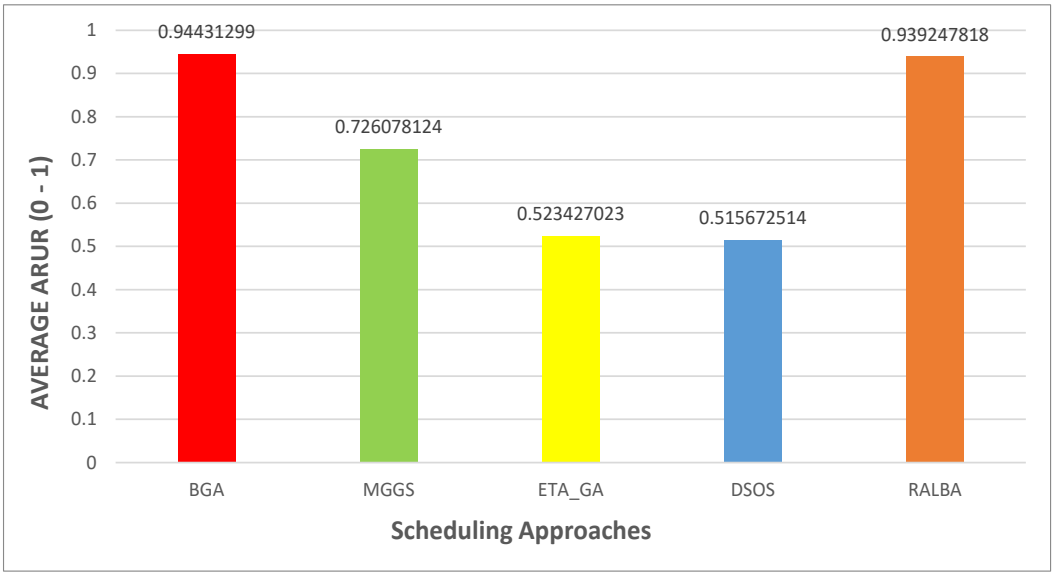
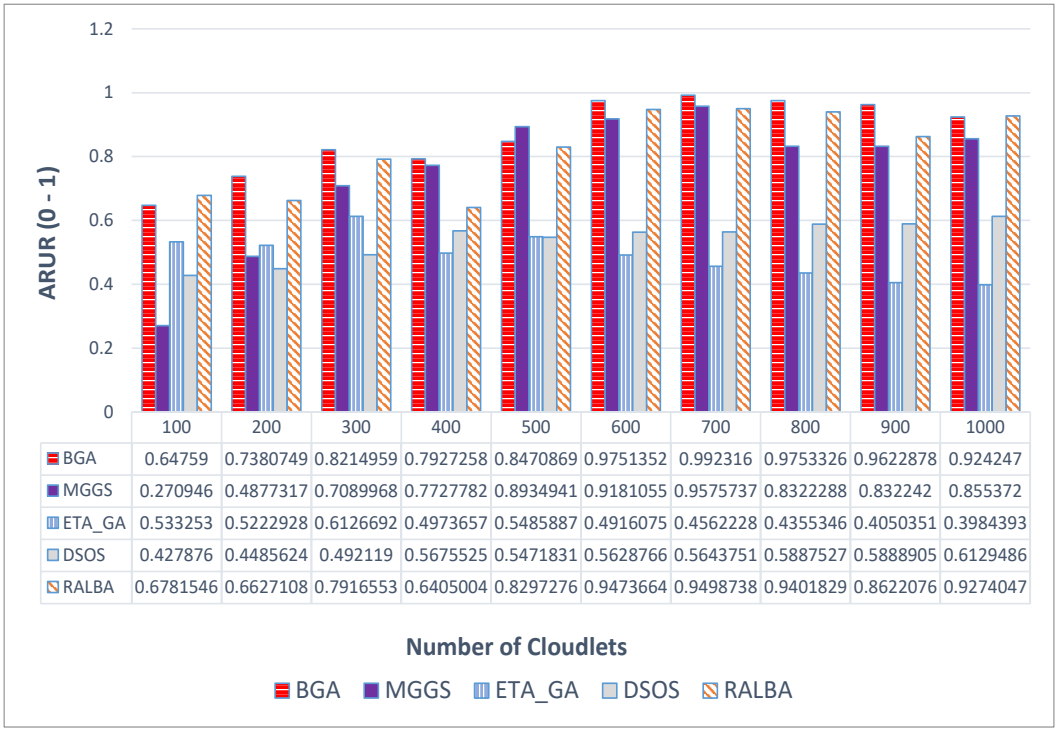


FIGURE 4.16: Average of ARUR on GoCJ Dataset

The analysis of BGA on left skewed dataset for ARUR reveals that BGA significantly improves the resource utilization on large batch sizes. The Figure 4.17 shows the ARUR on batch sizes 100 to 1000, whereas Figure 4.18 represents the average behavior on all batch sizes. From the average value it is evident that improvements in ARUR of BGA as compared to MGGS, ETA-GA, DSOS and

RALBA are 15.2, 77, 60.6, and 5.4% respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	0.64759	0.7380749	0.8214959	0.7927258	0.8470869	0.9751352	0.992316	0.9753326	0.9622878	0.924247
MGGS	0.270946	0.4877317	0.7089968	0.7727782	0.8934941	0.9181055	0.9575737	0.8322288	0.832242	0.855372
ETA_GA	0.533253	0.5222928	0.6126692	0.4973657	0.5485887	0.4916075	0.4562228	0.4355346	0.4050351	0.3984393
DSOS	0.427876	0.4485624	0.492119	0.5675525	0.5471831	0.5628766	0.5643751	0.5887527	0.5888905	0.6129486
RALBA	0.6781546	0.6627108	0.7916553	0.6405004	0.8297276	0.9473664	0.9498738	0.9401829	0.8622076	0.9274047

FIGURE 4.17: ARUR on Left Skewed Dataset

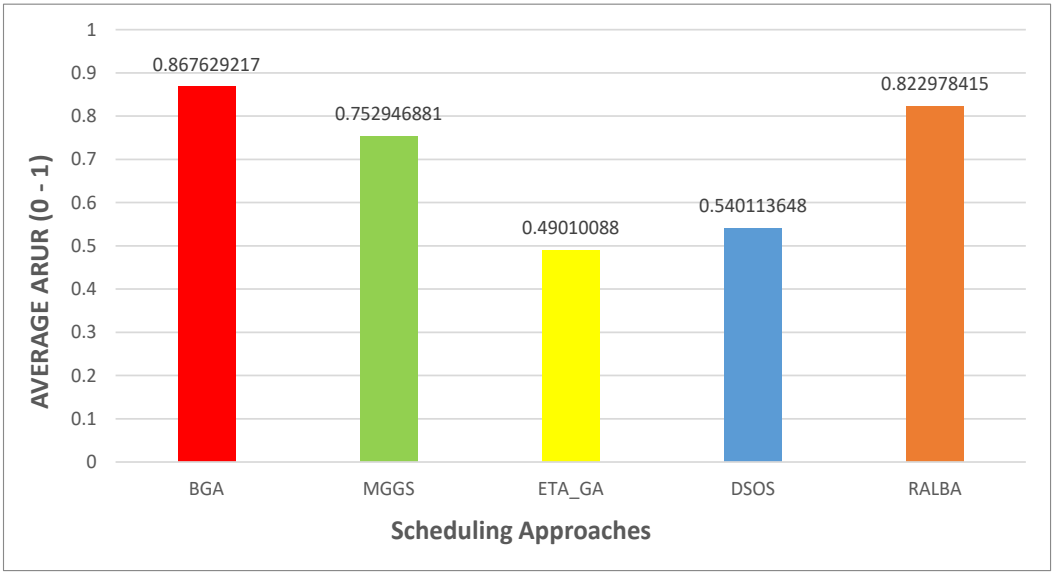
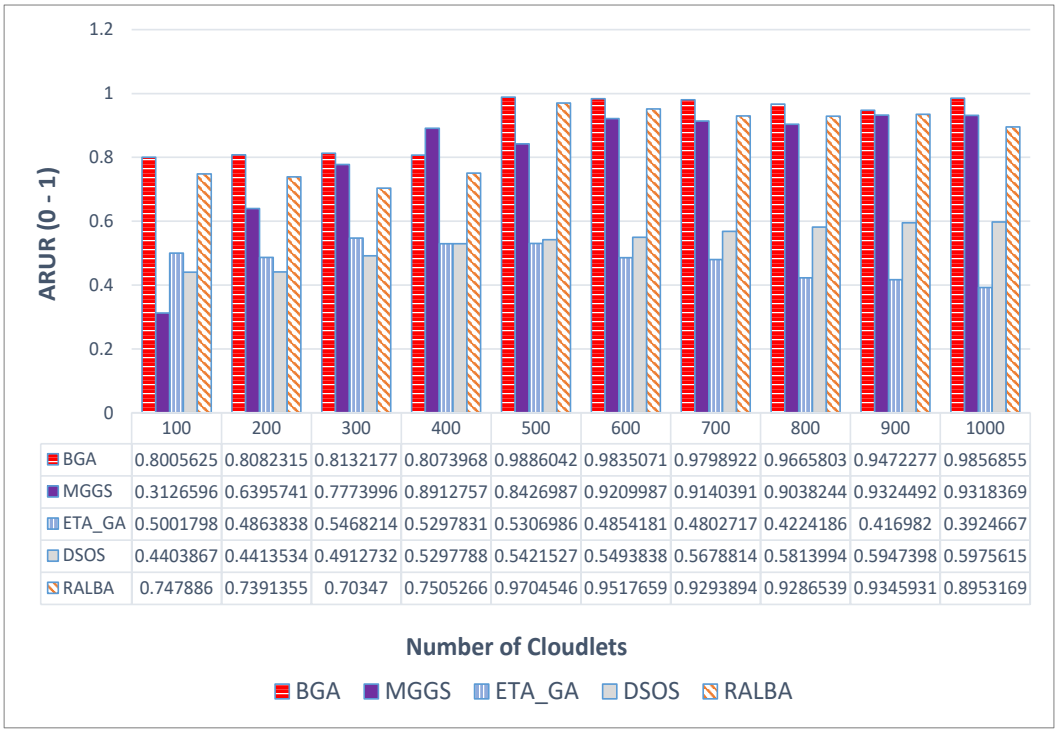


FIGURE 4.18: Average of ARUR on Left Skewed Dataset

The ARUR of BGA and other techniques is compared on right skewed dataset as shown in Figure 4.19 and the average of all batch sizes is exhibited in Figure 4.20. The BGA outperforms in most of the batch sizes whereas MGGS is better than RALBA in some large batch sizes. Overall on average the percentage of

improvement in ARUR of BGA is 12.5, 89.5, 70.1, and 6.1% higher than MGGS, ETA-GA, DSOS and RALBA respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	0.8005625	0.8082315	0.8132177	0.8073968	0.9886042	0.9835071	0.9798922	0.9665803	0.9472277	0.9856855
MGGS	0.3126596	0.6395741	0.7773996	0.8912757	0.8426987	0.9209987	0.9140391	0.9038244	0.9324492	0.9318369
ETA_GA	0.5001798	0.4863838	0.5468214	0.5297831	0.5306986	0.4854181	0.4802717	0.4224186	0.416982	0.3924667
DSOS	0.4403867	0.4413534	0.4912732	0.5297788	0.5421527	0.5493838	0.5678814	0.5813994	0.5947398	0.5975615
RALBA	0.747886	0.7391355	0.70347	0.7505266	0.9704546	0.9517659	0.9293894	0.9286539	0.9345931	0.8953169

FIGURE 4.19: ARUR on Right Skewed Dataset

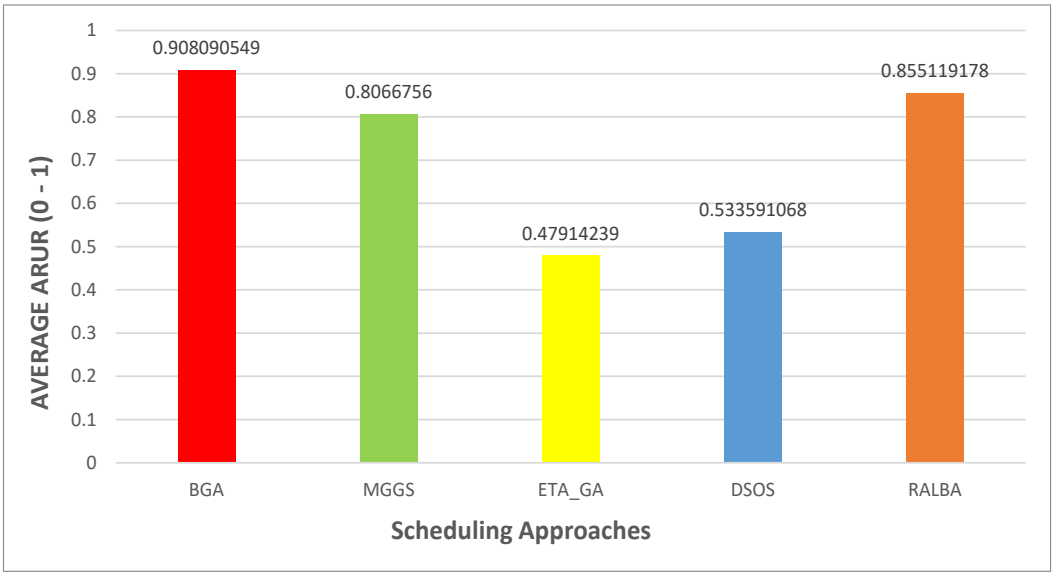
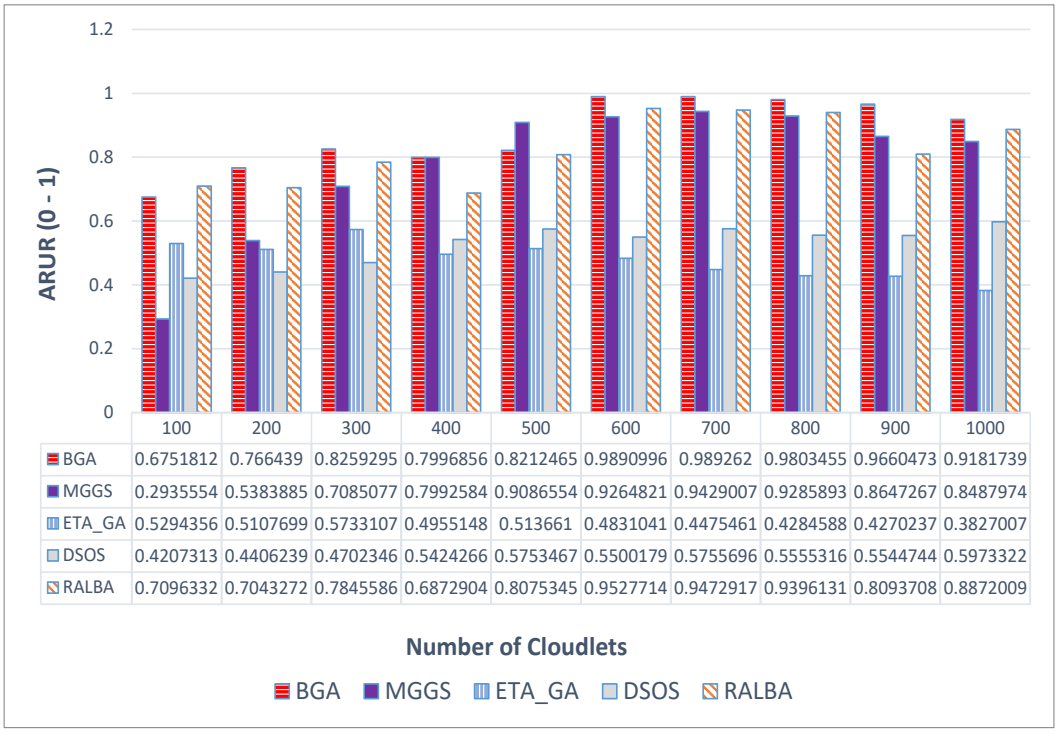


FIGURE 4.20: Average of ARUR on Right Skewed Dataset

The Figure 4.21 the performance of BGA is compared on normal dataset. It is inferred that BGA has shown good ARUR on large datasets and at batch size 500 MGGS is better. Mostly BGA is better whereas RALBA and MGGS are close

competitors in some cases. The Figure 4.22 shows the average behavior of all batch sizes. BGA improves 12.5, 82.2, 65.2, and 6% in ARUR as compared to MGGS, ETA-GA, DSOS and RALBA respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	0.6751812	0.766439	0.8259295	0.7996856	0.8212465	0.9890996	0.989262	0.9803455	0.9660473	0.9181739
MGGS	0.2935554	0.5383885	0.7085077	0.7992584	0.9086554	0.9264821	0.9429007	0.9285893	0.8647267	0.8487974
ETA_GA	0.5294356	0.5107699	0.5733107	0.4955148	0.513661	0.4831041	0.4475461	0.4284588	0.4270237	0.3827007
DSOS	0.4207313	0.4406239	0.4702346	0.5424266	0.5753467	0.5500179	0.5755696	0.5555316	0.5544744	0.5973322
RALBA	0.7096332	0.7043272	0.7845586	0.6872904	0.8075345	0.9527714	0.9472917	0.9396131	0.8093708	0.8872009

FIGURE 4.21: ARUR on Normal Dataset

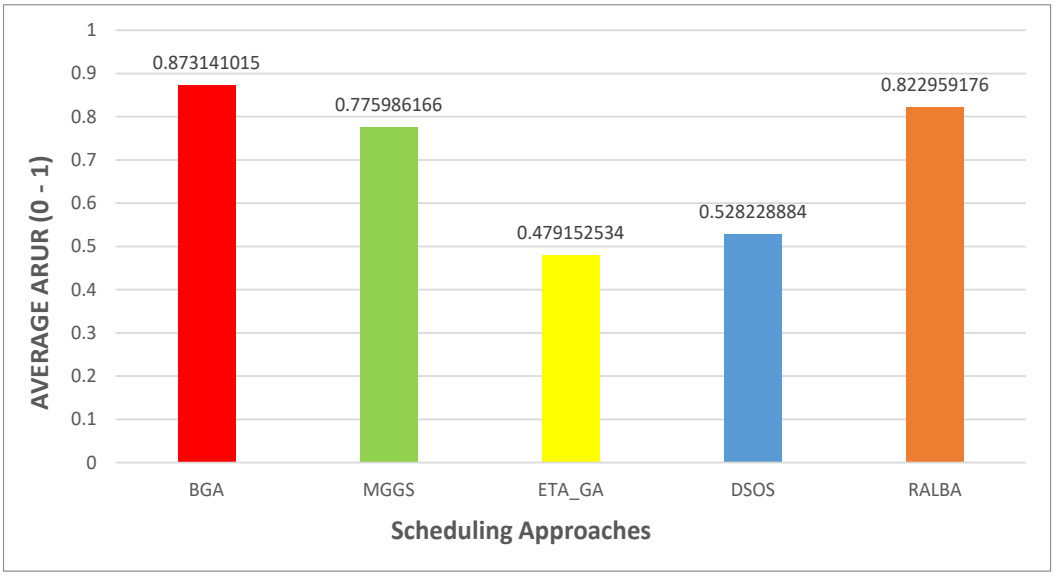
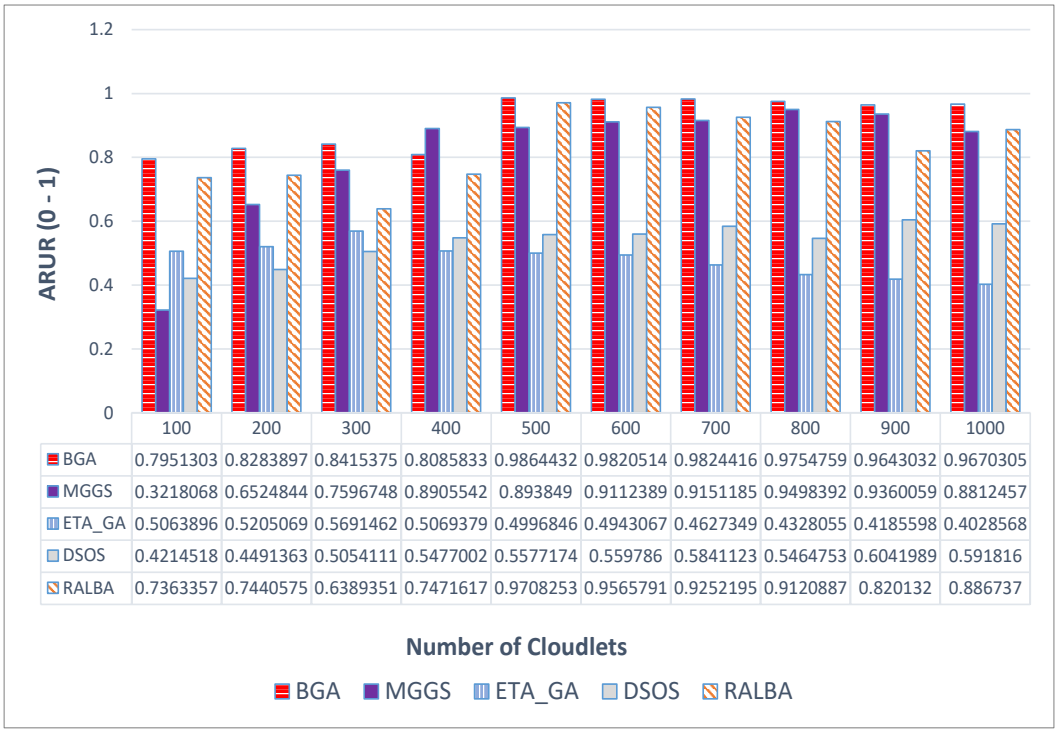


FIGURE 4.22: Average of ARUR on Normal Dataset

The BGA is compare on uniform dataset as shown in Figure 4.23. Mostly the value of ARUR is good using the BGA. The average ARUR of all batch sizes in

uniform dataset is shown in Figure 4.24. On average BGA is 12.5, 82.2, 65.2, and 6% better than MGGS, ETA-GA, DSOS and RALBA respectively.



	100	200	300	400	500	600	700	800	900	1000
BGA	0.7951303	0.8283897	0.8415375	0.8085833	0.9864432	0.9820514	0.9824416	0.9754759	0.9643032	0.9670305
MGGS	0.3218068	0.6524844	0.7596748	0.8905542	0.893849	0.9112389	0.9151185	0.9498392	0.9360059	0.8812457
ETA_GA	0.5063896	0.5205069	0.5691462	0.5069379	0.4996846	0.4943067	0.4627349	0.4328055	0.4185598	0.4028568
DSOS	0.4214518	0.4491363	0.5054111	0.5477002	0.5577174	0.559786	0.5841123	0.5464753	0.6041989	0.591816
RALBA	0.7363357	0.7440575	0.6389351	0.7471617	0.9708253	0.9565791	0.9252195	0.9120887	0.820132	0.886737

FIGURE 4.23: ARUR on Uniform Dataset

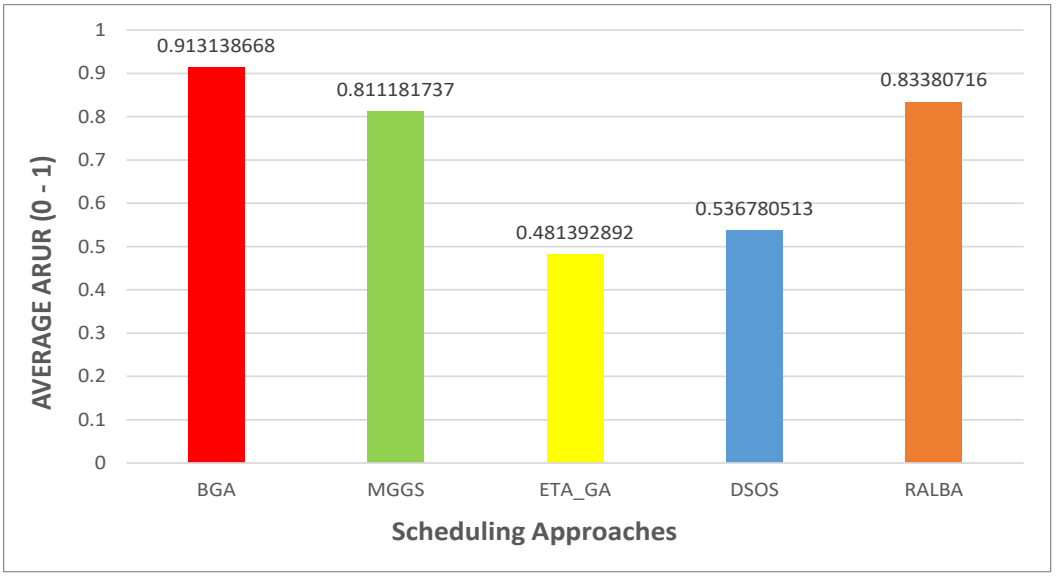


FIGURE 4.24: Average of ARUR on Uniform Dataset

Overall it can be concluded that BGA outperforms than other benchmark techniques in reducing the makespan while improving the load balancing. The research

question no.1 is to show how makespan and load balancing can be improved. Using the BGA the load balancing and makespan has improved in comparison to other state-of-the-art techniques. This has been demonstrated using extensive experimentation and findings are revealed in Section 4.2. The research question no.3 is about the effect of fusing heuristic with meta-heuristic in task scheduling. From the evaluation of BGA it can be said that the merger of heuristic with meta-heuristic assists in achieving the good fitness value for the defined objectives. The working of BGA has already been described in methodology chapter. The findings reveal that BGA balances the workload and improves the makespan, therefore the effect of fusion is positive.

4.3 SGA

The working of SGA has already been discussed in chapter no.3. SGA is applicable where meta-heuristic based scheduler is required and the main concern here is improvement in the convergence speed of GA. When to use SGA has already been discussed in Section 3.6. For convergence analysis only meta-heuristics can be compared with SGA. Usually the fitness value is compared and the fitness of properly implemented technique improves over the iterations. As the presented technique is multi-objective and the representation of fitness value is different from other techniques. Therefore, the techniques cannot be compared on the basis of fitness value. The fitness function in SGA is designed to improve the makespan and load balancing. The prime focus of SGA is to speed up the convergence of GA. To test the working of SGA with other techniques, makespan and ARUR are analyzed for 500 iterations. As SGA is claimed to be fast so 500 iterations are chosen to exhibit the behavior of convergence. Convergence does not really mean that SGA would not converge further after 500 iterations. The 500 iterations are chosen to show that SGA can reach to a good solution in fewer iterations. SGA does not have any heuristic embedded in it thus the working is just like a normal GA based meta-heuristic, except the speed of search process is fast.

SGA is compared with ETA-GA and DSOS as already discussed in Section 3.8. The value of makespan cannot decline over the iterations whereas the value of ARUR may decline. The fitness function is designed in such a way that it has makespan value added in the load balancing factor. The fitness function and the reasons for such a multi-objective relation is already described in chapter no.3. The makespan should be good and then for that good makespan value the best solution should be the one which has good load balancing. When a new best solution is found in SGA which has good makespan it is possible that the resource utilization is not as good as it was with previously found best solution. This is why the ARUR value may decline at some points in the search space. The convergence is about reaching to a good solution and fast convergence means quickly reaching to a good solution [10, 66, 73, 79]. This behavior of SGA is exhibited through experimentations. The experiments are conducted on two datasets for all batch sizes for 500 iterations.

The makespan and ARUR evaluated on realistic dataset with improvement is demonstrated in Table 4.1, 4.2, 4.3, 4.4, 4.5 for GoCJ, Left Skewed, Right Skewed, Normal and Uniform datasets respectively. The improvement in the makespan and ARUR value after 500 iterations shows that SGA converges fast to a good solution as compared to DSOS and ETA-GA. SGA keeps improving even after 500 iterations but initial 500 iterations are fast due to the mutualism phase.

TABLE 4.1: Makespan and ARUR on GoCJ Dataset

GoCJ Dataset	**Makespan**			**ARUR**		
Instances	**SGA**	**DSOS**	**ETA-GA**	**SGA**	**DSOS**	**ETA-GA**
100	324.9167	368.381	391.381	0.493056	0.411585	0.474886
150	463.7619	602.375	713.6667	0.651927	0.421811	0.533392
200	519.6667	770.3333	843.3333	0.749427	0.426566	0.535037
250	603.3333	800.3333	1045.333	0.744245	0.467848	0.546647
300	767.4286	1154.333	1338.333	0.835062	0.482277	0.580865
350	874.6667	1279.889	1743.333	0.820597	0.499872	0.550301
400	1012.705	1362.267	1934.333	0.815529	0.516789	0.549207
450	1015.4	1689.952	2347.333	0.863761	0.486778	0.514823
500	1340.667	1729.489	2576.889	0.820835	0.528553	0.54826
550	1419.667	1934.333	3268.333	0.844188	0.487583	0.47675
600	1711.556	2342.867	3491	0.83587	0.523907	0.53142
650	1781.222	2388.286	3754.667	0.852966	0.54233	0.526693
700	1769.889	2410.4	4439	0.854103	0.524951	0.470638
750	2077.389	2735.5	4876	0.84896	0.551245	0.490975
800	1997.333	2759.133	5046.667	0.860368	0.556561	0.478193
850	2150.444	2863.222	5810.667	0.857086	0.540586	0.461487
900	2319.833	3155.667	6261	0.875597	0.56886	0.472049
950	2394.333	3206.111	6763	0.872952	0.56943	0.44655
1000	2690.552	3349.3	7417.334	0.857144	0.589273	0.451763
Average	1433.409	1942.22	3371.663	0.808088	0.510358	0.507365

TABLE 4.2: Makespan and ARUR on Left Skewed Dataset

Left Skewed Dataset	**Makespan**			**ARUR**		
Instances	**SGA**	**DSOS**	**ETA-GA**	**SGA**	**DSOS**	**ETA-GA**
100	70.828	111.603	112.0033	0.608008	0.394794	0.478918
200	137.5129	231.5819	251.3087	0.709519	0.435346	0.499047
300	206.764	337.933	421.6253	0.742223	0.482385	0.561242
400	286.2604	415.8458	677.19	0.741419	0.558862	0.485646
500	374.587	527.3088	837.854	0.76762	0.541502	0.52116
600	441.5231	607.8028	1286.294	0.844841	0.545933	0.431756
700	533.948	708.4305	1478.967	0.827345	0.539684	0.437822
800	653.5373	784.06	2071.097	0.825268	0.586492	0.388065
900	696.4433	863.8748	2655.967	0.859334	0.59558	0.360534
1000	790.1988	971.4894	3056.184	0.852897	0.573807	0.354253
Average	419.1603	555.993	1284.849	0.777848	0.525439	0.451844

TABLE 4.3: Makespan and ARUR on Right Skewed Dataset

Right Skewed Dataset	**Makespan**			**ARUR**		
Instances	**SGA**	**DSOS**	**ETA-GA**	**SGA**	**DSOS**	**ETA-GA**
100	56.89543	95.54705	87.98	0.671826	0.421128	0.501459
200	111.4062	183.4893	223.4967	0.701454	0.438541	0.504708
300	168.5147	263.221	352.81	0.749591	0.501393	0.540513
400	240.6553	341.4451	495.9967	0.767312	0.543361	0.512504
500	289.364	414.9299	708.8464	0.858416	0.561273	0.499284
600	376.4221	528.7567	956.5007	0.820179	0.538311	0.471086
700	455.062	569.1076	1244.707	0.828156	0.564766	0.436286
800	510.564	656.81	1595.697	0.84787	0.586151	0.399604
900	565.812	726.9176	2013.59	0.854027	0.594033	0.372961
1000	642.4182	800.6078	2204.27	0.851923	0.595052	0.380695
Average	341.7114	458.0832	988.3894	0.795075	0.534401	0.46191

TABLE 4.4: Makespan and ARUR on Normal Dataset

Normal Dataset	**Makespan**			**ARUR**		
Instances	**SGA**	**DSOS**	**ETA-GA**	**SGA**	**DSOS**	**ETA-GA**
100	77.35533	121.7691	120.1471	0.602952	0.407817	0.501366
200	147.7476	232.8318	287.5244	0.714275	0.440364	0.490894
300	228.6627	372.0059	439.6753	0.734346	0.486339	0.562949
400	310.9887	456.7033	722.548	0.739946	0.54788	0.504662
500	384.6657	553.4755	995.2967	0.764508	0.580643	0.472532
600	472.5077	676.7886	1327.797	0.846837	0.526342	0.45595
700	607.4657	763.464	1645.899	0.808521	0.550871	0.438681
800	692.9867	857.08	2081.943	0.83081	0.600624	0.412057
900	749.09	962.3262	2542.92	0.861566	0.559447	0.388602
1000	886.2303	1062.875	3087.51	0.837535	0.590557	0.364594
Average	455.77	605.932	1325.126	0.77413	0.529088	0.459229

TABLE 4.5: Makespan and ARUR on Uniform Dataset

Uniform Dataset	**Makespan**			**ARUR**		
Instances	**SGA**	**DSOS**	**ETA-GA**	**SGA**	**DSOS**	**ETA-GA**
100	72.199	126.7525	118.54	0.64739	0.413737	0.497374
200	138.5447	234.1715	268.4493	0.724816	0.449463	0.5033
300	210.2592	326.2798	464.1067	0.737971	0.467321	0.511934
400	288.9	420.9528	677.8533	0.767379	0.548463	0.487874
500	358.144	523.5364	870.5531	0.853099	0.552948	0.506337
600	463.1556	618.09	1219.887	0.821739	0.5793	0.461766
700	564.6367	720.7766	1630.27	0.833491	0.561443	0.427475
800	638.4333	842.1916	1936.103	0.862735	0.582226	0.419837
900	714.3951	889.5197	2406.237	0.848897	0.588171	0.388018
1000	802.1932	994.7424	2817.136	0.84965	0.574878	0.369923
Average	425.086	569.7013	1240.913	0.794717	0.531795	0.457384

To exhibit the behavior of fast convergence the performance measures are demonstrated for all datasets. The convergence of performance measures on batch size of 500 and 1000 are expressed for 500 iterations.

4.3.1 Makespan

The Figure 4.25 shows the convergence for the makespan value on batch of 500 tasks using GoCJ dataset. It shows that SGA is fast in convergence than DSOS and ETA-GA. The Figure 4.26 shows the makespan on batch size 1000 of GoCJ dataset. On large size of batch, the ETA-GA converges very slow whereas SGA is fastest among all.

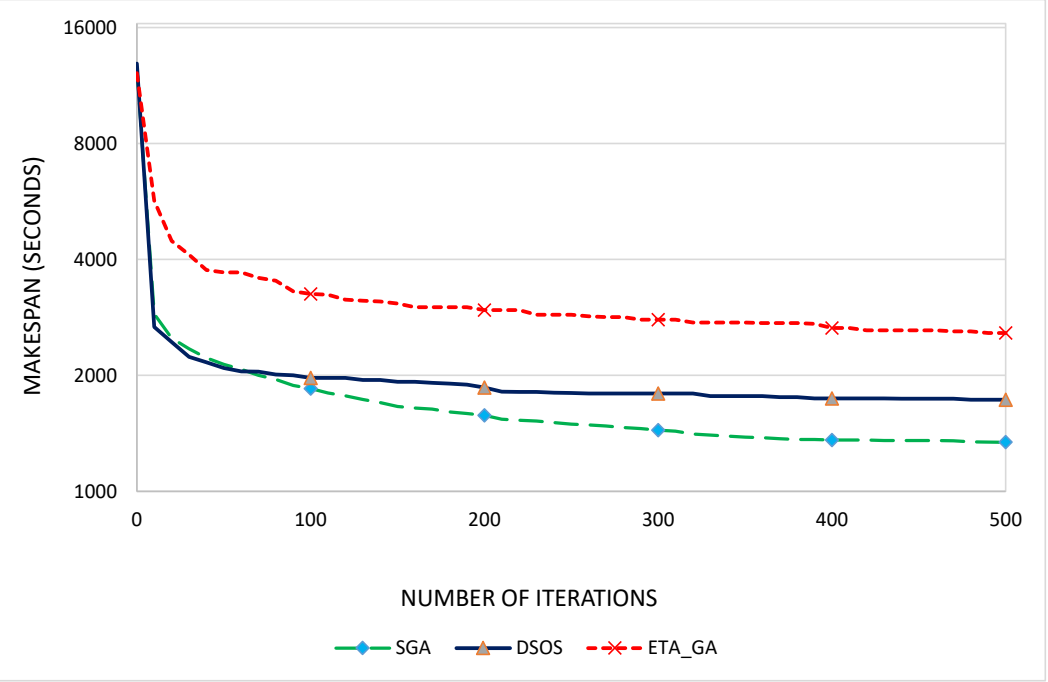


FIGURE 4.25: Makespan on GoCJ Dataset of 500 Jobs

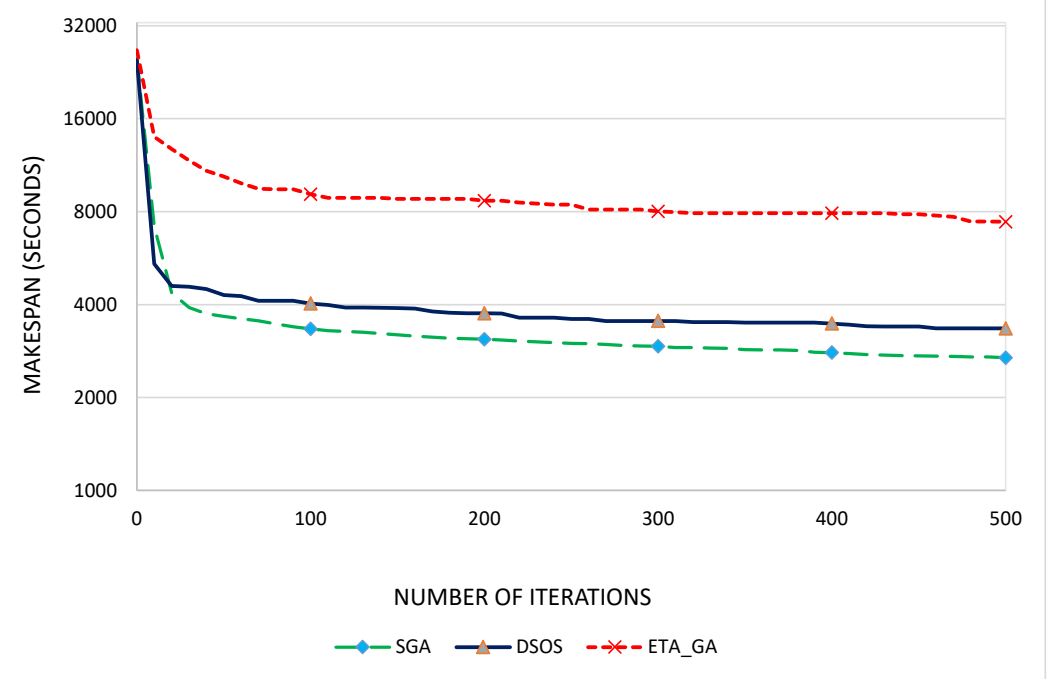


FIGURE 4.26: Makespan on GoCJ Dataset of 1000 Jobs

The Figure 4.27 and Figure 4.28 shows the convergence of SGA and other techniques on Left Skewed dataset for batch of 500 and 1000 respectively. On left skewed dataset SGA is fastest among all.

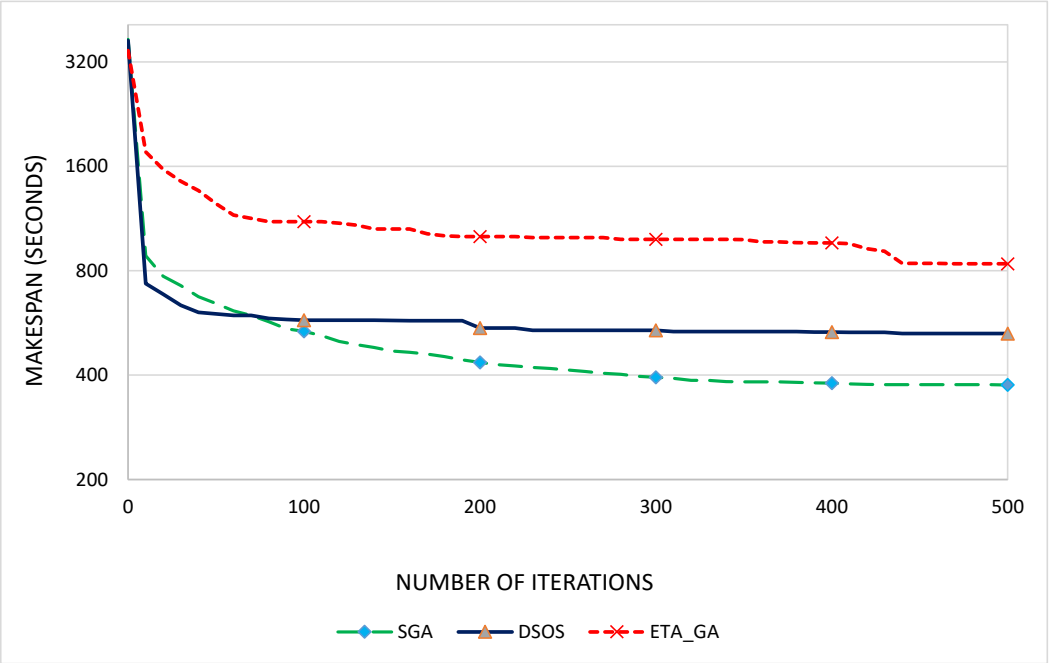


FIGURE 4.27: Makespan on Left Skewed Dataset of 500 Jobs

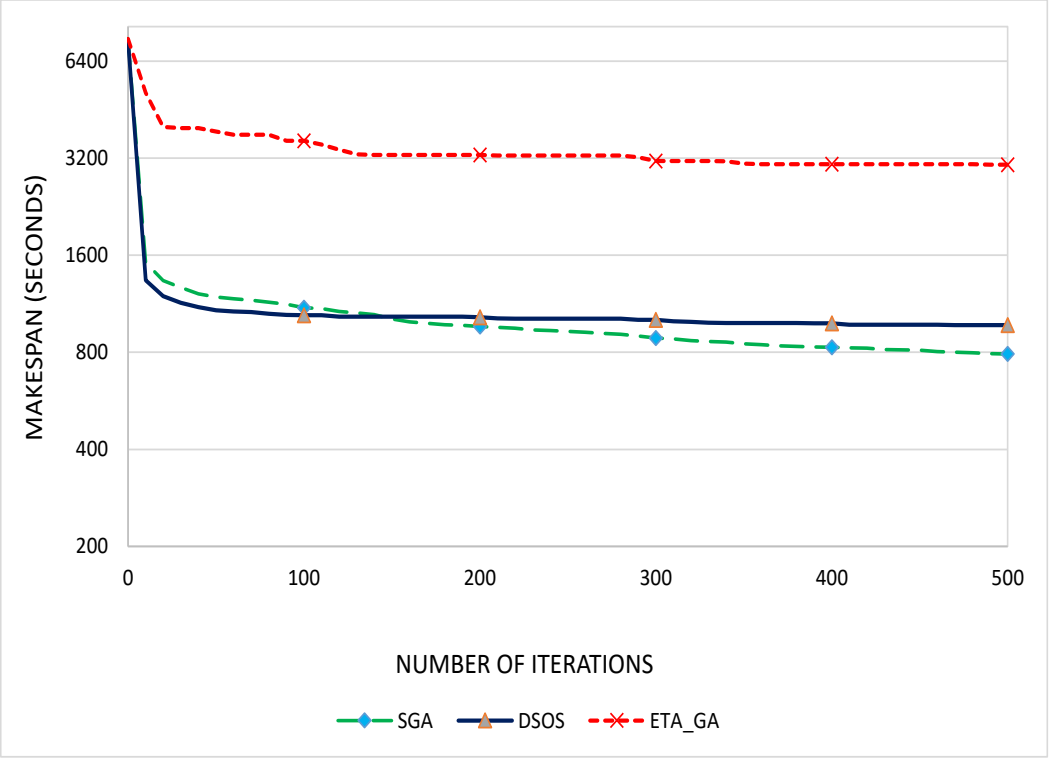


FIGURE 4.28: Makespan on Left Skewed Dataset of 1000 Jobs

The Figure 4.29 and 4.30 shows the convergence on right skewed dataset for batch size of 500 and 1000 respectively. Here DSOS is comparatively close than ETA-GA but SGA outperforms.

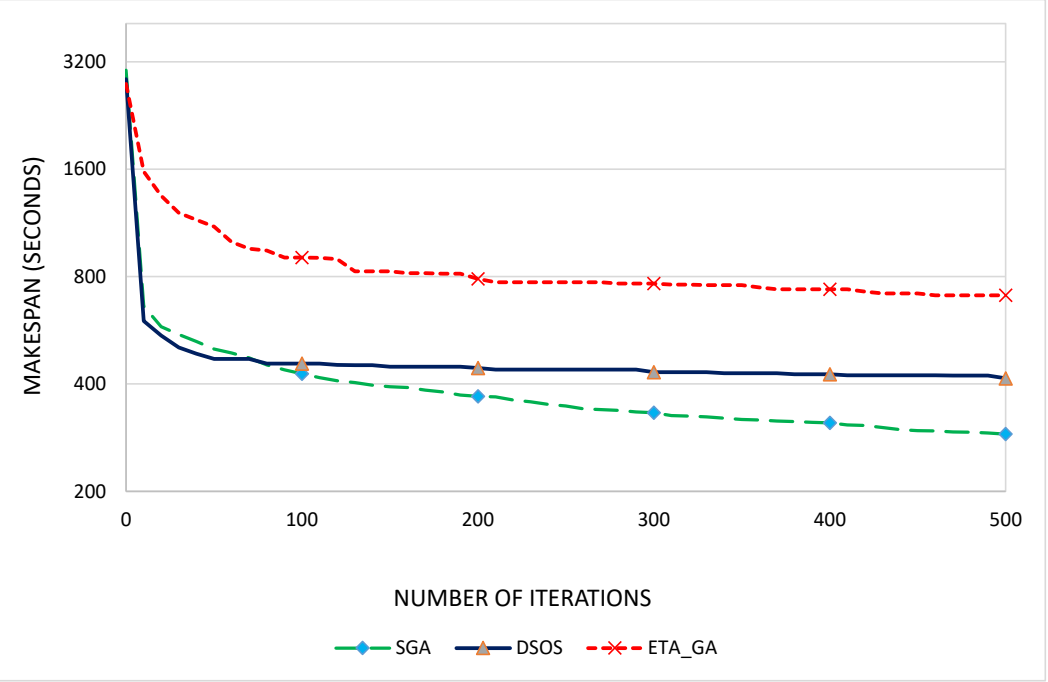


FIGURE 4.29: Makespan on Right Skewed Dataset of 500 Jobs

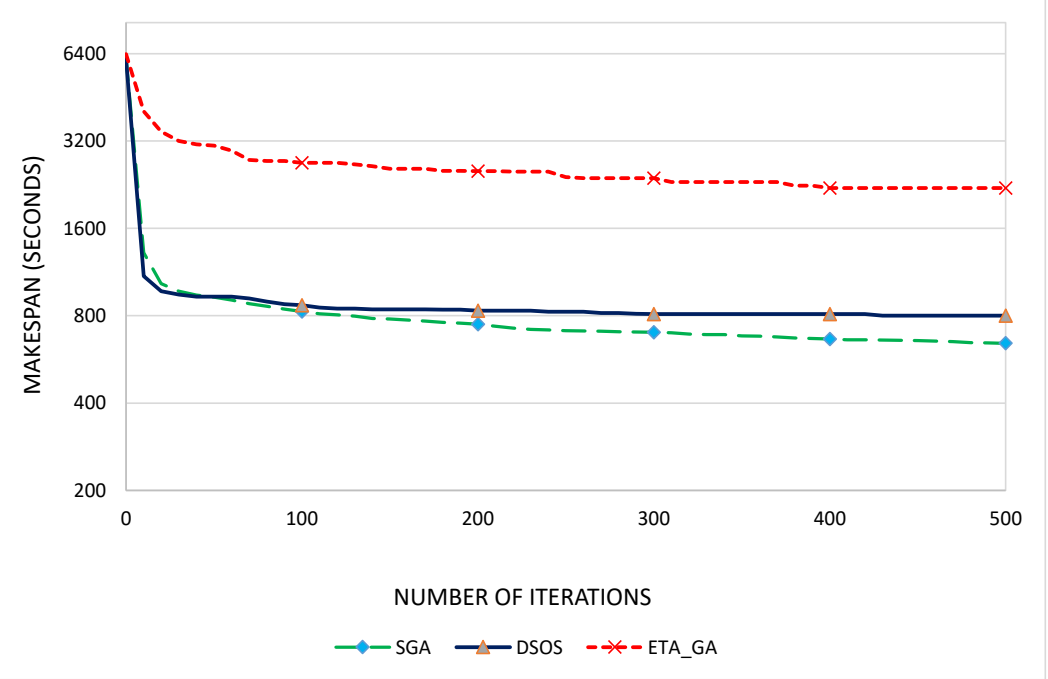


FIGURE 4.30: Makespan on Right Skewed Dataset of 1000 Jobs

On normal dataset at batch 500 and 1000 the ETA-GA improves but still lack in speed than SGA and DSOS, whereas DSOS converges faster than all. It is shown in Figure 4.31 and 4.32.

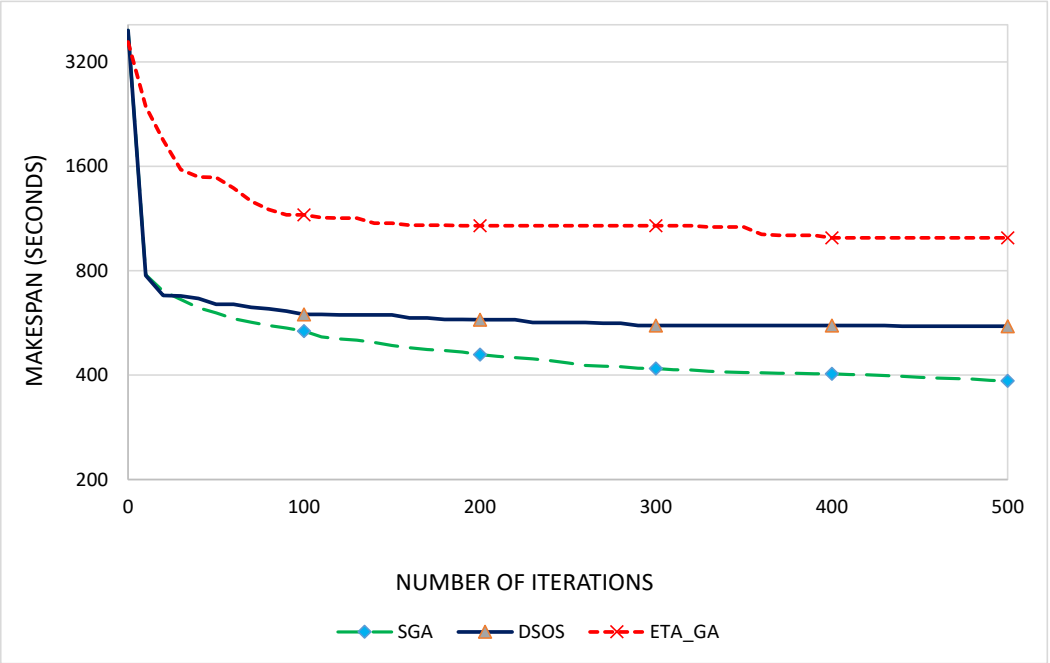


FIGURE 4.31: Makespan on Normal Dataset of 500 Jobs

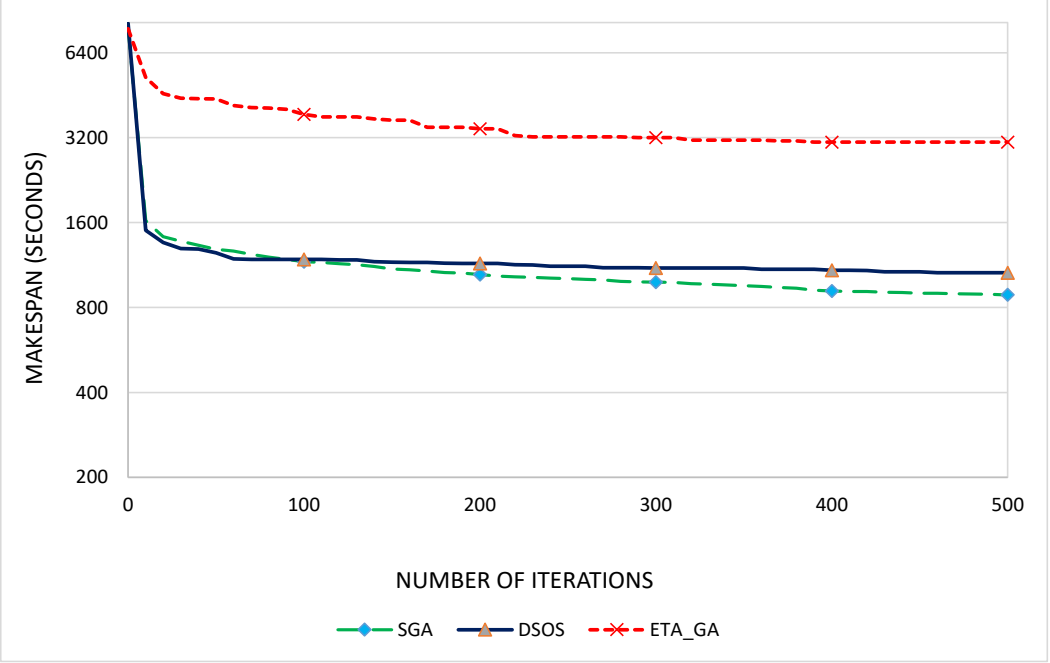


FIGURE 4.32: Makespan on Normal Dataset of 1000 Jobs

The convergence on uniform dataset is demonstrated in Figure 4.33 and 4.34 for batch of 500 and 1000 respectively. It is evident that SGA outperforms in convergence speed for all datasets.

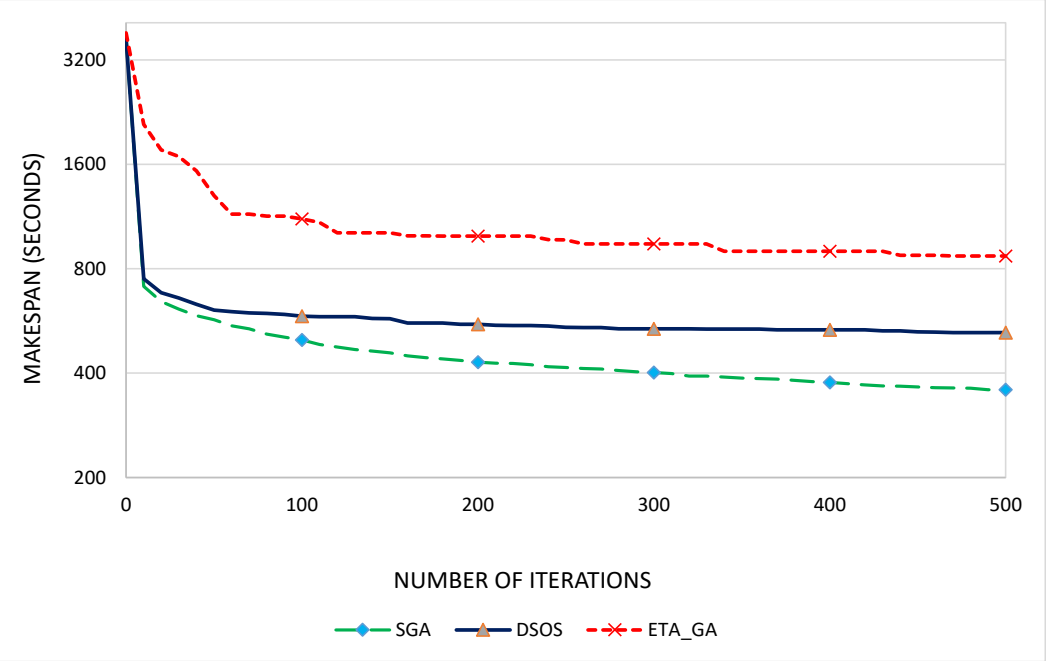


FIGURE 4.33: Makespan on Uniform Dataset of 500 Jobs

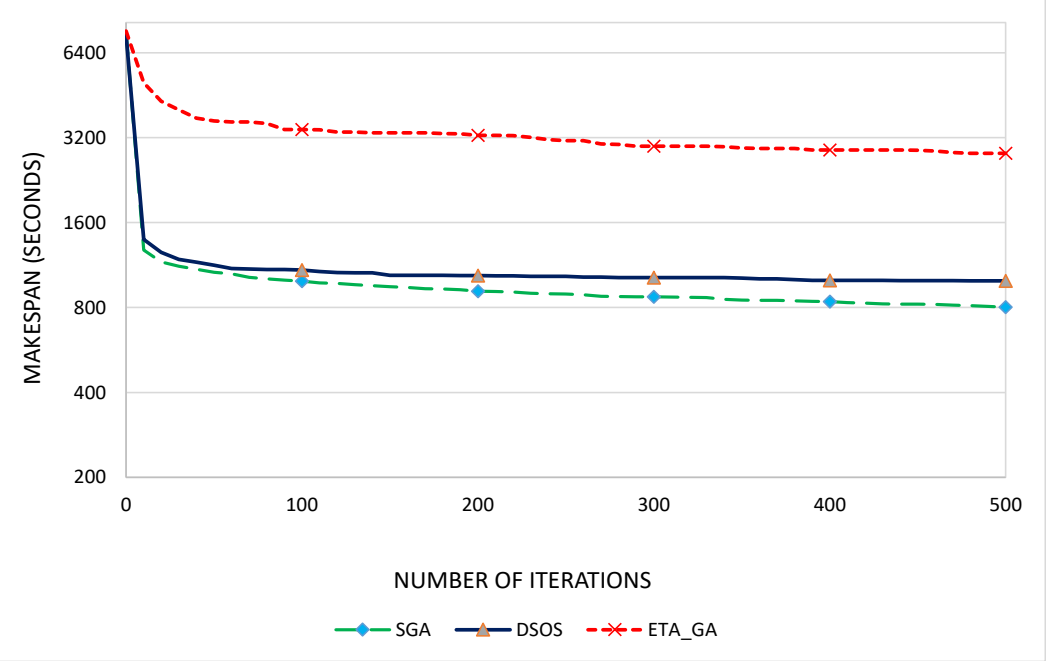


FIGURE 4.34: Makespan on Uniform Dataset of 1000 Jobs

4.3.2 ARUR

The Figure 4.35 and 4.36 shows the analysis of ARUR on GoCJ dataset for batch of 500 and 1000 respectively. SGA significantly outperform for ARUR.

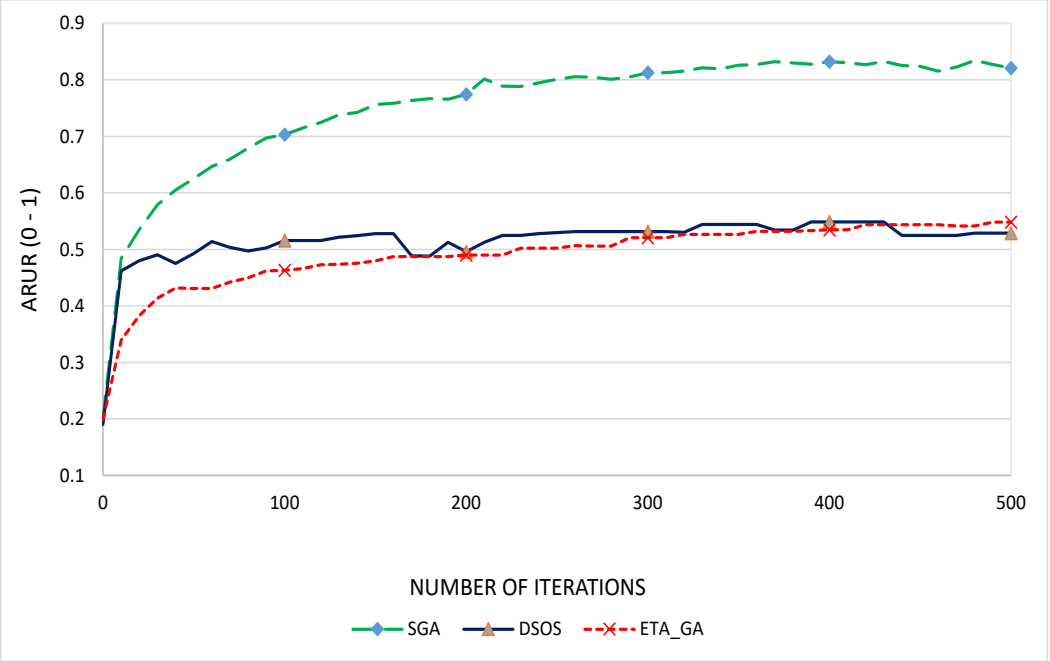


FIGURE 4.35: ARUR on GoCJ Dataset of 500 Jobs

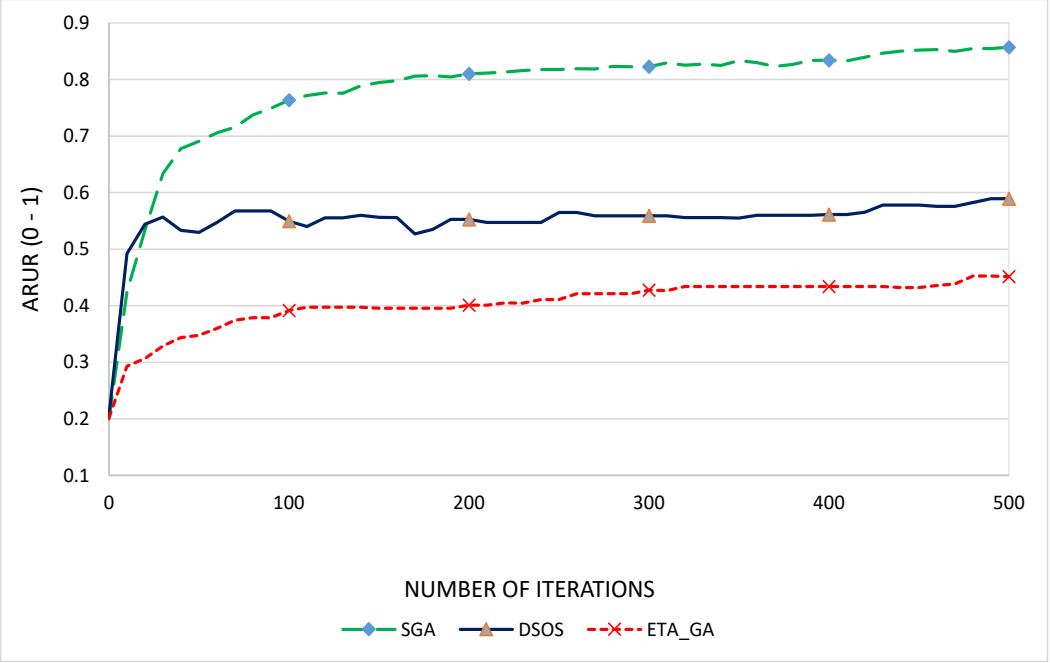


FIGURE 4.36: ARUR on GoCJ Dataset of 1000 Jobs

The Figure 4.37 shows the convergence on left skewed dataset for ARUR value on batch of 500 whereas the Figure 4.38 shows convergence on batch of 1000. The SGA outperforms here.

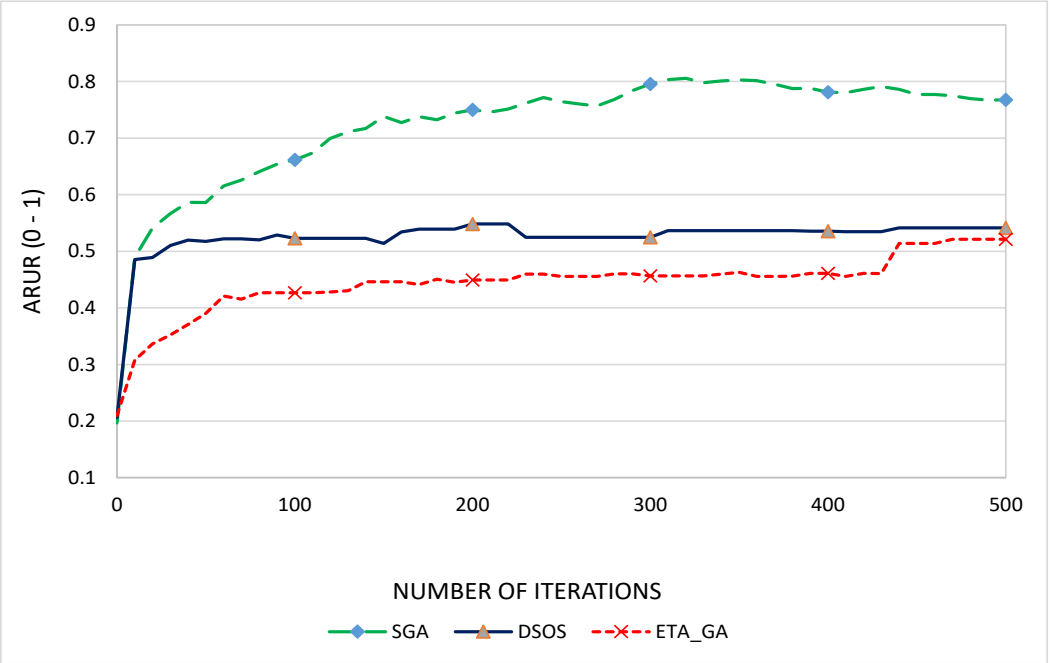


FIGURE 4.37: ARUR on Left Skewed Dataset of 500 Jobs

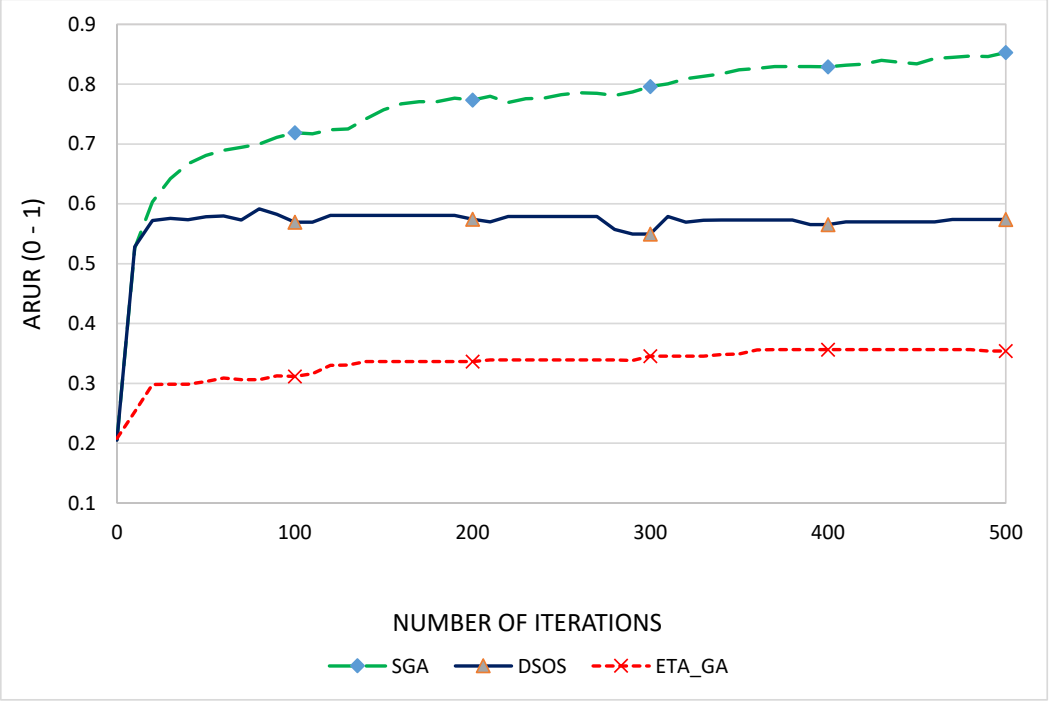


FIGURE 4.38: ARUR on Left Skewed Dataset of 1000 Jobs

Figure 4.39 and 4.40 shows convergence on right skewed dataset for ARUR value on batch of 500 and 1000 respectively. Here SGA outperforms than other techniques.

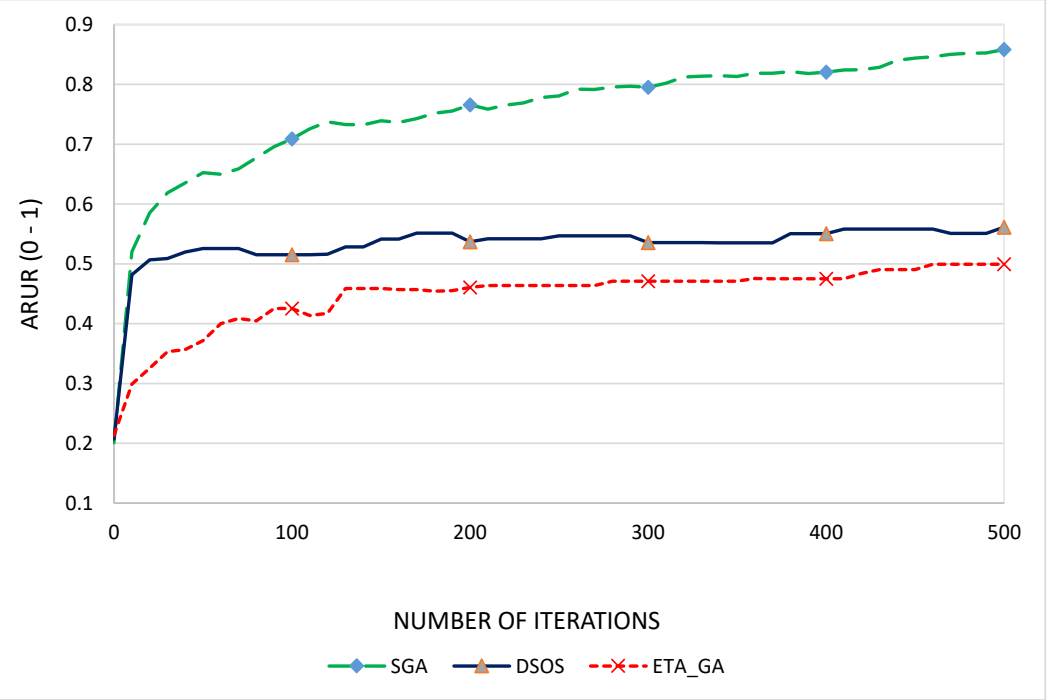


FIGURE 4.39: ARUR on Right Skewed Dataset of 500 Jobs

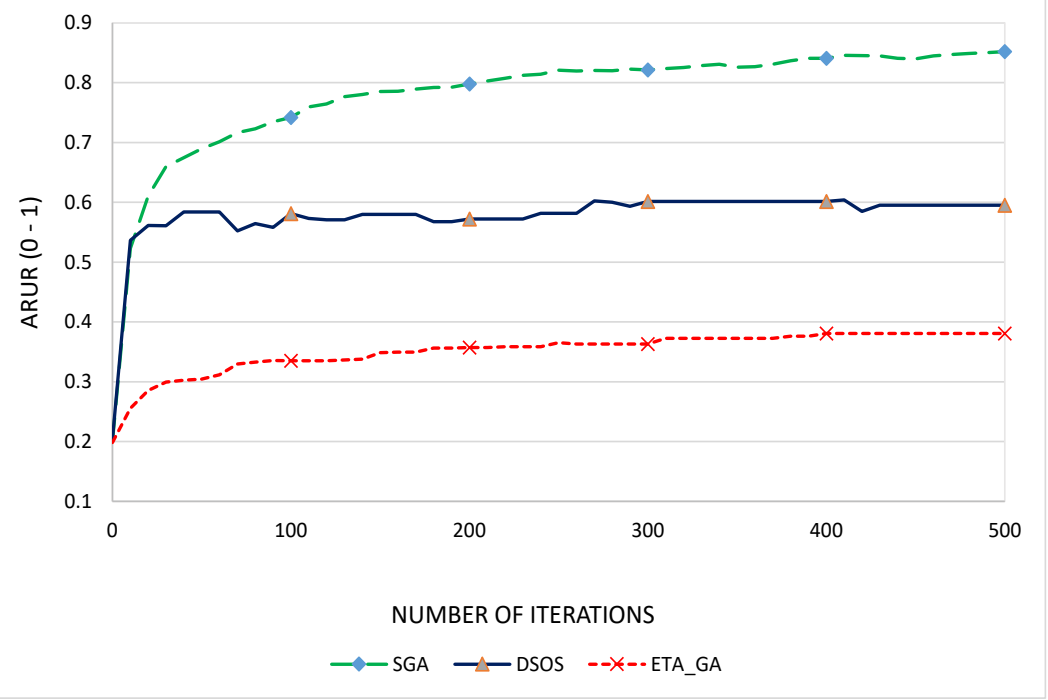


FIGURE 4.40: ARUR on Right Skewed Dataset of 1000 Jobs

On normal dataset for batch of 500 and 100 the SGA converges faster than other techniques and it is shown in Figure 4.41 and 4.42.

On the uniform dataset at batch size 500 the SGA outperforms in ARUR but

FIGURE 4.41: ARUR on Normal Dataset of 500 Jobs

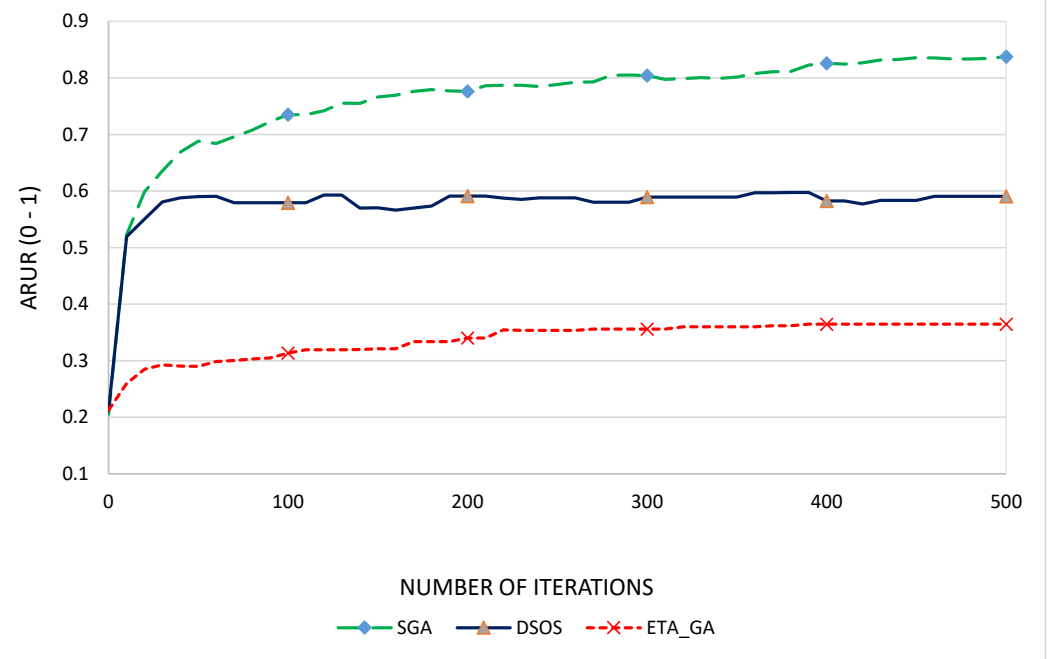


FIGURE 4.42: ARUR on Normal Dataset of 1000 Jobs

DSOS and ETA-GA are relatively close to each other. In batch size 1000 the difference of ARUR value is large and again the SGA converges faster. It is shown in Figure 4.43 and 4.44.

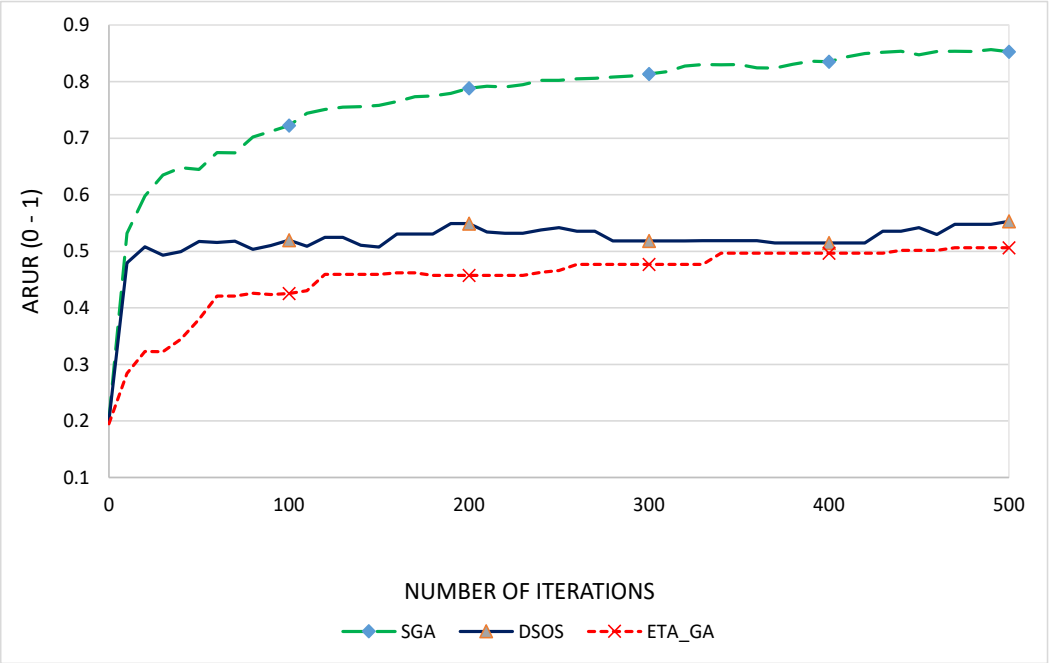


FIGURE 4.43: ARUR on Uniform Dataset of 500 Jobs

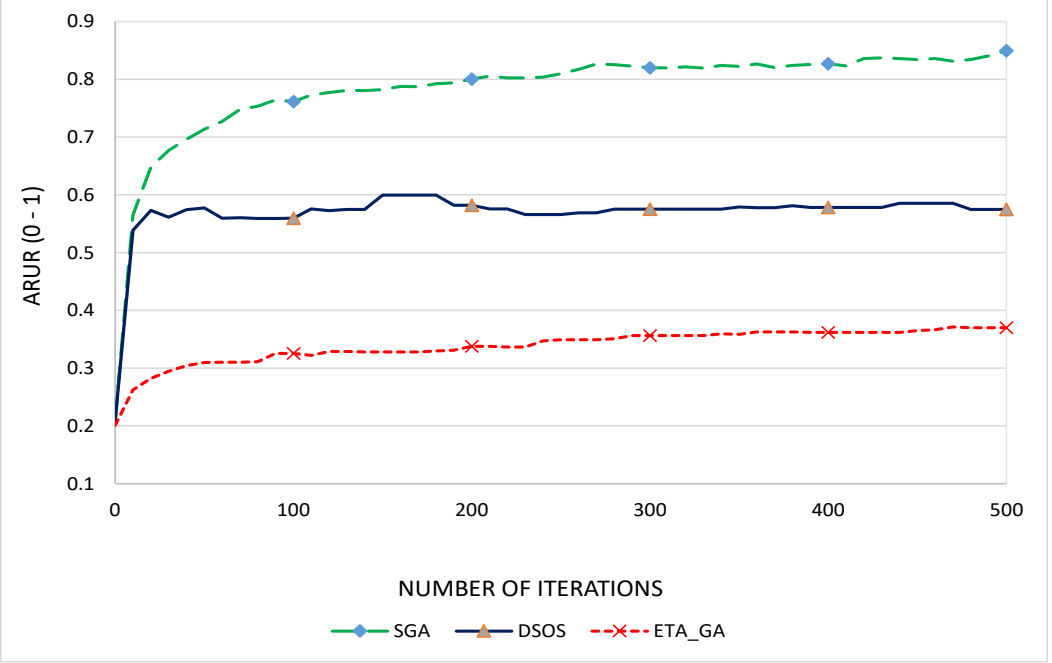


FIGURE 4.44: ARUR on Uniform Dataset of 1000 Jobs

From the experimentations it is apparent that BGA and SGA performs better in achieving their objectives which are mapped with the research questions and the mapping is expressed in Section 3.6.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The task scheduling problem in cloud computing requires the efficient mapping of jobs to virtual resources. Due to the heterogeneity of jobs and resources many possible mappings can be defined. The heuristic and meta-heuristic schedulers are utilized to map independent jobs. The meta-heuristic has potential to explore the huge search space of possible solutions. The genetic based BGA has been presented in the research to improve the makespan and resource utilization. Whereas, the SGA also focuses on the convergence speed. The presented techniques are compared with some recent meta-heuristic and heuristic techniques. The genetic based other techniques are also evaluated with the presented techniques. The synthetic and realistic datasets are employed to discover the working of the presented techniques. The need for multi-objective optimization is emphasized and the presented work demonstrates the techniques for achieving multi-objectives. Following facets are reflected in the research work.

1. The problem of independent task scheduling in heterogeneous cloud environment is emphasized.
2. Heuristic, meta-heuristic and hybrid techniques are meticulously evaluated for optimal scheduling.

3. Need for the adoptability of meta-heuristic is rigorously accentuated.

4. The power of genetic based evolutionary approach in meta-heuristic is rationalized.

5. Two most prominent quality of service (QoS) parameters namely makespan and resource utilization are concentrated in this thesis for the definition of optimal mapping.

6. For the achievement of enhancement in the said QoS, the multi-objective optimization is accentuated as necessity and a relation in the form of objective function is articulated.

7. Fine tuning of parameters in BGA and SGA is detailed for balancing the exploration and exploitation process.

8. A load balancing mechanism is presented and deployed through BGA for improving makespan in a load balanced way.

9. The symbiosis relationship is inculcated in GA to form SGA, to improve the convergence of genetic based task scheduler.

10. The proportionate selection operator with scaling is projected to improve diversity in the population of the presented schedulers.

11. The performance is evaluated with state-of-the-art techniques on two different types of dataset.

5.2 Future Work

Cloud computing environment has both dependent and independent jobs. The presented work only focuses on the independent task scheduling whereas in future the presented techniques can be extended to comply with the requirements of dependent jobs. The scheduling can be static or dynamic. The presented work is for static task scheduling. Static and dynamic scheduling have their own advantages and applications. In future the same concepts of presented work with some necessary changes can also be used for dynamic scheduling. Cloud computing has lot of challenges and one prime objective is to improve the customer satisfaction. The demand of quality of services introduces the concept of service

level agreement (SLA) in which the services are negotiated among service consumer and provider. Currently the presented work does not take in consideration the priorities or SLA. The presented work may also be enhanced to invigorate the SLA fulfillment through the scheduler. There are many possible amendments experimented in the meta-heuristic presented approach and successful are deployed. There are still many prospects and some of the directions for improvement in genetic based task scheduler are expressed in the below section.

1. Finding the effect of mutation after the mutualism phase.
2. Generating new population with symbiotic operator rather than the crossover.
3. Experimenting with different number of elites and variable crossover rate.
4. Defining a rule based adaptive mutation rate in genetic algorithm.
5. Using the concept of distance of best chromosome as reference point in the crossover operation.
6. Using different population initialization strategies in task scheduling.
7. Examining the effect of having more than two cut-points of crossover on the population.
8. Merging GA with other popular meta-heuristics.
9. Defining weights for over and underloaded machines to have better load balancing using BGA.
10. Dedicating a portion of population to be generated by less fitted chromosomes so that their genes may also participate in each generation rather than in some random generations of GA.

Bibliography

[1] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," *International Conference on Advanced Information Networking and Applications*, vol. 8, no. 1, pp. 27–33, 2010.

[2] F. Yiqiu, X. Xia, and G. Junwei, "Cloud computing task scheduling algorithm based on improved genetic algorithm," *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 7, no. 4, pp. 852–856, 2019.

[3] C. H. . G. Karagiannis, "Cloud computing services: taxonomy and comparison," *J Internet Serv Appl*, vol. 2, no. 2, pp. 81–94, 2011.

[4] M. Abdullahi, M. A. Ngadi, S. I. Dishing, B. I. Ahmad *et al.*, "An efficient symbiotic organisms search algorithm with chaotic optimization strategy for multi-objective task scheduling problems in cloud computing environment," *Journal of Network and Computer Applications*, vol. 133, no. 1, pp. 60–74, 2019.

[5] M. Abdullahi, M. A. Ngadi *et al.*, "Symbiotic organism search optimization based task scheduling in cloud computing environment," *Future Generation Computer Systems*, vol. 56, no. 1, pp. 640–650, 2016.

[6] M. Reza Mesbahi, A. Masoud Rahmani, and M. Hosseinzadeh, "Reliability and high availability in cloud computing environments: a reference roadmap," *Human Centric Computing and Information Sciences*, vol. 3, no. 5, pp. 101–131, 2018.

[7] A. Hussain, M. Aleem, A. Khan, M. A. Iqbal, and M. A. Islam, "Ralba: a computation-aware load balancing scheduler for cloud computing," *Cluster Computing*, vol. 21, no. 3, pp. 1667–1680, 2018.

[8] K. Duan, S. Fong, S. W. Siu, W. Song, and S. S.-U. Guan, "Adaptive incremental genetic algorithm for task scheduling in cloud environments," *Symmetry*, vol. 10, no. 5, pp. 168–180, 2018.

[9] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong, "The characteristics of cloud computing," *International Conference on Parallel Processing Workshops*, vol. 7, no. 3, pp. 275–279, 2010.

[10] T. S. Alnusairi, A. A. Shahin, and Y. Daadaa, "Binary psogsa for load balancing task scheduling in cloud environment," *arXiv preprint arXiv:1806.00329*, vol. 1, no. 5, pp. 255–264, 2018.

[11] M. Abdullahi and M. A. Ngadi, "Hybrid symbiotic organisms search optimization algorithm for scheduling of tasks on cloud computing environment," *PloS one*, vol. 11, no. 6, pp. 229–257, 2016.

[12] Y. Xing and Y. Zhan, "Virtualization and cloud computing," *Future Wireless Networks and Information Systems*, vol. 12, no. 6, pp. 305–312, 2012.

[13] A. Hussain, M. Aleem, M. A. Iqbal, and M. A. Islam, "Sla-ralba: cost-efficient and resource-aware load balancing algorithm for cloud computing," *The Journal of Supercomputing*, vol. 75, no. 10, pp. 6777–6803, 2019.

[14] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy, and M. U. Chowdhury, "An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments," *Neural Computing and Applications*, vol. 2, no. 3, pp. 1–11, 2019.

[15] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorihtm for grid task scheduling," *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*, vol. 1, no. 2, pp. 1–7, 2007.

[16] T. Kokilavani, D. G. Amalarethinam *et al.*, "Load balanced min-min algorithm for static meta-task scheduling in grid computing," *International Journal of Computer Applications*, vol. 20, no. 2, pp. 43–49, 2011.

[17] S. Kaur and A. Verma, "An efficient approach to genetic algorithm for task scheduling in cloud computing environment," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 4, no. 10, pp. 74–79, 2012.

[18] Z. Mohamad, A. A. Mahmoud, W. N. S. W. Nik, M. A. Mohamed, and M. M. Deris, "A genetic algorithm for optimal job scheduling and load balancing in cloud computing," *International Journal of Engineering & Technology*, vol. 7, no. 3.28, pp. 290–294, 2018.

[19] S. A. Hamad and F. A. Omara, "Genetic-based task scheduling algorithm in cloud computing environment," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 4, pp. 550–556, 2016.

[20] S. Choe, B. Li, I. Ri, C. Paek, J. Rim, and S. Yun, "Improved hybrid symbiotic organism search task-scheduling algorithm for cloud computing," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 12, no. 8, pp. 3516–3541, 2018.

[21] V. A. Xavier and S. Annadurai, "Chaotic social spider algorithm for load balance aware task scheduling in cloud computing," *Cluster Computing*, vol. 22, no. 1, pp. 287–297, 2019.

[22] S. K. Panda and P. K. Jana, "An energy-efficient task scheduling algorithm for heterogeneous cloud computing systems," *Cluster Computing*, vol. 22, no. 2, pp. 509–527, 2019.

[23] S. Kaur and J. Sengupta, "Load balancing using improved genetic algorithm (iga) in cloud computing," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 6, no. 8, pp. 2278–1323, 2017.

[24] Z.-H. Zhan, G.-Y. Zhang, Y.-J. Gong, J. Zhang *et al.*, "Load balance aware genetic algorithm for task scheduling in cloud computing," *Asia-Pacific Conference on Simulated Evolution and Learning*, vol. 2, no. 1, pp. 644–655, 2014.

[25] A. A. Beegom and M. Rajasree, "Integer-pso: a discrete pso algorithm for task scheduling in cloud computing systems," *Evolutionary Intelligence*, vol. 12, no. 2, pp. 227–239, 2019.

[26] R. Masadeh, A. Sharieh, and B. Mahafzah, "Humpback whale optimization algorithm based on vocal behavior for task scheduling in cloud computing," *Int J Adv Sci Technol*, vol. 13, no. 3, pp. 121–140, 2019.

[27] S. H. H. Madni, M. S. A. Latiff, J. Ali *et al.*, "Hybrid gradient descent cuckoo search (hgdcs) algorithm for resource scheduling in iaas cloud computing environment," *Cluster Computing*, vol. 22, no. 1, pp. 301–334, 2019.

[28] K. Sreenu and M. Sreelatha, "W-scheduler: whale optimization for task scheduling in cloud computing," *Cluster Computing*, vol. 7, no. 1, pp. 1–12, 2017.

[29] N. Almezeini and A. Hafez, "Task scheduling in cloud computing using lion optimization algorithm," *Algorithms*, vol. 5, no. 1, pp. 77–83, 2017.

[30] F. Ebadifard and S. M. Babamir, "A pso-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 12, pp. 4368–4383, 2018.

[31] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian informatics journal*, vol. 16, no. 3, pp. 275–295, 2015.

[32] Z. Xu, X. Xu, and X. Zhao, "Task scheduling based on multi-objective genetic algorithm in cloud computing," *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, vol. 12, no. 4, pp. 1429–1438, 2015.

[33] S. K. Panda and P. K. Jana, "Load balanced task scheduling for cloud computing: A probabilistic approach," *Knowledge and Information Systems*, vol. 61, no. 3, pp. 1607–1631, 2019.

[34] N. Mansouri, B. M. H. Zade, and M. M. Javidi, "Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory," *Computers & Industrial Engineering*, vol. 130, no. 1, pp. 597–633, 2019.

[35] A. Hussain, M. Aleem, M. A. Islam, and M. Iqbal, "A rigorous evaluation of state-of-the-art scheduling algorithms for cloud computing," *IEEE Access*, vol. 6, no. 1, pp. 75 033–75 047, 2018.

[36] A. A. Nasr, N. A. El-Bahnasawy, G. Attiya, and A. El-Sayed, "Using the tsp solution strategy for cloudlet scheduling in cloud computing," *Journal of Network and Systems Management*, vol. 27, no. 2, pp. 366–387, 2019.

[37] H. Krishnaveni and V. S. J. Prakash, "Execution time based sufferage algorithm for static task scheduling in cloud," *Advances in Big Data and Cloud Computing*, vol. 6, no. 2, pp. 61–70, 2019.

[38] M. Habibi and N. J. Navimipour, "Multi-objective task scheduling in cloud computing using an imperialist competitive algorithm," *Int J Adv Comput Sci Appl*, vol. 1, no. 7, pp. 289–293, 2016.

[39] Z. Zhong, K. Chen, X. Zhai, and S. Zhou, "Virtual machine-based task scheduling algorithm in a cloud computing environment," *Tsinghua Science and Technology*, vol. 21, no. 6, pp. 660–667, 2016.

[40] P. Jaglan and C. Diwakar, "Partical swarm optimization of task scheduling in cloud computing," *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES and RESEARCH TECHNOLOGY*, vol. 21, no. 6, pp. 660–667, 2016.

[41] K. B. Bey, F. Benhammadi, and R. Benaissa, "Balancing heuristic for independent task scheduling in cloud computing," *2015 12th International Symposium on Programming and Systems (ISPS)*, vol. 5, no. 3, pp. 1–6, 2015.

[42] M. R. Abid, K. Kaddouri, M. D. El Ouadghiri, and D. Benhaddou, "Virtual machines' migration for cloud computing," *CLOUD COMPUTING*, vol. 107, no. 1, pp. 97–102, 2018.

[43] S. Kaur and P. Pandey, "A survey of virtual machine migration techniques in cloud computing," *Computer Engineering and Intelligent Systems*, vol. 6, no. 7, pp. 28–34, 2015.

[44] K. Sudha and S. Sukumaran, "Coherent genetic algorithm for task scheduling in cloud computing environment," *Aust. J. Basic Appl. Sci*, vol. 9, no. 2, pp. 1–8, 2015.

[45] A. Vohra and Suchitra, "Deadline-aware sufferage based task scheduling in cloud computing," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no. 5, pp. 9526–9532, 2017.

[46] N. Thakkar and R. Nath, "Discrete artificial bee colony algorithm for load balancing in cloud computing environment," *International Journal of P2P Network Trends and Technology*, vol. 8, no. 6, pp. 101–107, 2018.

[47] X. Li, J. Song, and B. Huang, "A scientific workflow management system architecture and its scheduling based on cloud service platform for manufacturing big data analytics," *The International Journal of Advanced Manufacturing Technology*, vol. 84, no. 1-4, pp. 119–131, 2015.

[48] M. Kumara and S.C.Sharmab, "Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing," *The International Journal of Advanced Manufacturing Technology*, vol. 115, no. 1, pp. 322–329, 2017.

[49] N. Thakkar and R. Nath, "Performance analysis of min-min, max-min and artificial bee colony load balancing algorithms in cloud computing," *International Journal of Innovations and Advancement in Computer Science*, vol. 7, no. 4, pp. 185–191, 2018.

[50] G. Megharaj and M. Kabadi, "Hybrid model for load balancing and server consolidation in cloud," *International Journal of Intelligent Engineering and Systems*, vol. 12, no. 1, pp. 174–188, 2019.

[51] M. Kumar and S. C. Sharma, "Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment," *International Journal of Computers and Applications*, vol. 115, no. 1, pp. 322–329, 2017.

[52] P. Rekha and M. Dakshayini, "Efficient task allocation approach using genetic algorithm for cloud environment," *Cluster Computing*, vol. 22, no. 4, pp. 1241–1251, 2019.

[53] T. Wang, Z. Liu, Y. Chen, Y. Xu, and X. Dai, "Load balancing task scheduling based on genetic algorithm in cloud computing," *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, vol. 12, no. 3, pp. 146–152, 2014.

[54] A. S. Kumar and M. Venkatesan, "Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment," *Wireless Personal Communications*, vol. 107, no. 4, pp. 1835–1848, 2019.

[55] S. Javanmardi, M. Shojafar, D. Amendola, N. Cordeschi, H. Liu, and A. Abraham, "Hybrid job scheduling algorithm for cloud computing environment," *Proceedings of the fifth international conference on innovations in bio-inspired computing and applications IBICA 2014*, vol. 5, no. 2, pp. 43–52, 2014.

[56] S. Singh and M. Kalra, "Scheduling of independent tasks in cloud computing using modified genetic algorithm," *2014 International Conference on Computational Intelligence and Communication Networks*, vol. 3, no. 4, pp. 565–569, 2014.

[57] K. KRISHNASAMY *et al.*, "Task scheduling algorithm based on hybrid particle swarm optimization in cloud computing environment." *Journal of Theoretical & Applied Information Technology*, vol. 55, no. 1, pp. 33–38, 2013.

[58] S. Zhao, X. Fu, H. Li, G. Dong, and J. Li, "Research on cloud computing task scheduling based on improved particle swarm optimization," *International Journal of Performability Engineering*, vol. 13, no. 7, pp. 1063–1069, 2017.

[59] A. A. Beegom and M. Rajasree, "A particle swarm optimization based pareto optimal task scheduling in cloud computing," *International Conference in Swarm Intelligence*, vol. 6, no. 5, pp. 79–86, 2014.

[60] F. Ramezani, J. Lu, and F. K. Hussain, "Task-based system load balancing in cloud computing using particle swarm optimization," *International journal of parallel programming*, vol. 42, no. 5, pp. 739–754, 2014.

[61] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu, "Independent tasks scheduling based on genetic algorithm in cloud computing," *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, vol. 13, no. 4, pp. 1–4, 2009.

[62] X. Wu, M. Deng, R. Zhang, B. Zeng, and S. Zhou, "A task scheduling algorithm based on qos-driven in cloud computing," *Procedia Computer Science*, vol. 17, no. 1, pp. 1162–1169, 2013.

[63] T. Deepa and D. Cheelu, "A comparative study of static and dynamic load balancing algorithms in cloud computing," *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, vol. 9, no. 3, pp. 3375–3378, 2017.

[64] V. Manglani, A. Jain, and V. Prasad, "Task scheduling in cloud computing," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 3, pp. 821–825, 2018.

[65] T. P. Jacob and K. Pradeep, "A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization," *Wireless Personal Communications*, vol. 109, no. 1, pp. 315–331, 2019.

[66] A. Al-Maamari and F. A. Omara, "Task scheduling using pso algorithm in cloud computing environments," *International Journal of Grid and Distributed Computing*, vol. 8, no. 5, pp. 245–256, 2015.

[67] S. Abdi, S. A. Motamedi, and S. Sharifian, "Task scheduling using modified pso algorithm in cloud computing environment," *International conference on machine learning, electrical and mechanical engineering*, vol. 4, no. 1, pp. 8–12, 2014.

[68] K. Maryam, M. Sardaraz, and M. Tahir, "Evolutionary algorithms in cloud computing from the perspective of energy consumption: A review," *2018 14th International Conference on Emerging Technologies (ICET)*, vol. 4, no. 12, pp. 1–6, 2018.

[69] M. A. Elaziz, S. Xiong, K. Jayasena, and L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," *Knowledge-Based Systems*, vol. 169, no. 1, pp. 39–52, 2019.

[70] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[71] I. Strumberger, M. Tuba, N. Bacanin, and E. Tuba, "Cloudlet scheduling by hybridized monarch butterfly optimization algorithm," *Journal of Sensor and Actuator Networks*, vol. 8, no. 3, pp. 44–82, 2019.

[72] D. Gabi, A. S. Ismail, A. Zainal, Z. Zakaria, and A. Abraham, "Orthogonal taguchi-based cat algorithm for solving task scheduling problem in cloud computing," *Neural Computing and Applications*, vol. 30, no. 6, pp. 1845–1863, 2018.

[73] S. Zhan and H. Huo, "Improved pso-based task scheduling algorithm in cloud computing," *Journal of Information & Computational Science*, vol. 9, no. 13, pp. 3821–3829, 2012.

[74] X. He, X. Sun, and G. Von Laszewski, "Qos guided min-min heuristic for grid task scheduling," *Journal of computer science and technology*, vol. 18, no. 4, pp. 442–451, 2003.

[75] K. Pradeep and T. P. Jacob, "Cgsa scheduler: A multi-objective-based hybrid approach for task scheduling in cloud environment," *Information Security Journal: A Global Perspective*, vol. 27, no. 2, pp. 77–91, 2018.

[76] X. Huang, C. Li, H. Chen, and D. An, "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies," *Cluster Computing*, vol. 3, no. 6, pp. 1–11, 2019.

[77] A. Al-Qerem and A. Hamarsheh, "Statistical-based heuristic for tasks scheduling in cloud computing environment," *International Journal of Communication Networks and Information Security*, vol. 10, no. 2, pp. 358–365, 2018.

[78] S. Tareghian and Z. BORNAEE, "A new approach for scheduling jobs in cloud computing environment," *Cumhuriyet Üniversitesi Fen-Edebiyat Fakültesi Fen Bilimleri Dergisi*, vol. 36, no. 3, pp. 2499–2506, 2015.

[79] G. Kaur and E. S. Sharma, "Optimized utilization of resources using improved particle swarm optimization based task scheduling algorithms in cloud computing," *International Journal of Emerging Technology and Advanced Engineering*, vol. 4, no. 6, pp. 110–115, 2014.

[80] Y. Mao, X. Chen, and X. Li, "Max–min task scheduling algorithm for load balance in cloud computing," *Proceedings of International Conference on Computer Science and Information Technology*, vol. 3, no. 5, pp. 457–465, 2014.

[81] J.-T. Tsai, J.-C. Fang, and J.-H. Chou, "Optimized task scheduling and resource allocation on cloud computing environment using improved differential

evolution algorithm," *Computers & Operations Research*, vol. 40, no. 12, pp. 3045–3055, 2013.

[82] F. Ramezani, J. Lu, and F. Hussain, "Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization," *International Conference on Service-oriented computing*, vol. 4, no. 2, pp. 237–251, 2013.

[83] U. Bhoi, P. N. Ramanuj *et al.*, "Enhanced max-min task scheduling algorithm in cloud computing," *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, vol. 2, no. 4, pp. 259–264, 2013.

[84] O. Elzeki, M. Reshad, and M. Elsoud, "Improved max-min algorithm in cloud computing," *International Journal of Computer Applications*, vol. 50, no. 12, pp. 22–27, 2012.

[85] S. Bitam, "Bees life algorithm for job scheduling in cloud computing," *Proceedings of The Third International Conference on Communications and Information Technology*, vol. 7, no. 8, pp. 186–191, 2012.

[86] H. Saleh, H. Nashaat, W. Saber, and H. M. Harb, "Ipso task scheduling algorithm for large scale data in cloud computing environment," *IEEE Access*, vol. 7, no. 1, pp. 5412–5420, 2018.

[87] V. Lo, J. Mache, and K. Windisch, "A comparative study of real workload traces and synthetic workload models for parallel job scheduling," *Workshop on job scheduling strategies for parallel processing*, vol. 2, no. 2, pp. 25–46, 1998.

[88] N. Sharma, S. Tyagi, and S. Atri, "A comparative analysis of min-min and max-min algorithms based on the makespan parameter." *International Journal of Advanced Research in Computer Science*, vol. 8, no. 3, pp. 1038–1041, 2017.

[89] A. Hussain and M. Aleem, "Gocj: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, pp. 38–49, 2018.

[90] Y. Li, S. Wang, X. Hong, and Y. Li, "Multi-objective task scheduling optimization in cloud computing based on genetic algorithm and differential evolution algorithm," *2018 37th Chinese Control Conference (CCC)*, vol. 2, no. 2, pp. 4489–4494, 2018.

[91] S. Singh and M. Kalra, "Scheduling of independent tasks in cloud computing using modified genetic algorithm," *2014 International Conference on Computational Intelligence and Communication Networks*, vol. 12, no. 1, pp. 565–569, 2014.

[92] S. Kumar and M. Kalra, "A hybrid approach for energy-efficient task scheduling in cloud," *Proceedings of 2nd International Conference on Communication, Computing and Networking*, vol. 5, no. 1, pp. 1011–1019, 2019.

[93] A. M. Manasrah and H. Ba Ali, "Workflow scheduling using hybrid ga-pso algorithm in cloud computing," *Wireless Communications and Mobile Computing*, vol. 18, no. 1, pp. 1–16, 2018.

[94] B. M. Nguyen, H. Thi Thanh Binh, B. Do Son *et al.*, "Evolutionary algorithms to optimize task scheduling problem for the iot based bag-of-tasks application in cloud–fog computing environment," *Applied Sciences*, vol. 9, no. 9, pp. 1730–1749, 2019.

[95] M. Abdullahi, M. A. Ngadi, and S. I. Dishing, "Chaotic symbiotic organisms search for task scheduling optimization on cloud computing environment," *2017 6th ICT International Student Project Conference (ICT-ISPC)*, vol. 3, no. 2, pp. 1–4, 2017.

[96] S. Kumar, S. Mittal, and M. Singh, "A comparative study of metaheuristics based task scheduling in distributed environment," *Indian Journal of Science and Technology*, vol. 10, no. 26, pp. 100–112, 2017.