HYBRID ANT COLONY SYSTEM ALGORITHM FOR STATIC AND DYNAMIC JOB SCHEDULING IN GRID COMPUTING

MUSTAFA MUWAFAK THEAB ALOBAEDY

DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2015

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Pengkomputeran grid adalah sistem teragih dengan infrastruktur heterogen. Sistem pengurusan sumber (RMS) adalah salah satu komponen terpenting yang mempunyai pengaruh besar ke atas prestasi pengkomputeran grid. Bahagian utama RMS adalah algoritma penjadual yang bertanggungjawab untuk memeta tugas kepada sumber sedia ada. Kerumitan masalah penjadualan dianggap sebagai satu masalah lengkap polinomial tidak berketentuan (NP-lengkap) dan oleh itu, satu algoritma pintar diperlukan untuk mencapai penyelesaian penjadualan yang lebih baik. Salah satu algoritma pintar yang menonjol adalah ant colony system (ACS) yang digunakan secara meluas untuk menyelesaikan pelbagai jenis masalah penjadualan. Walau bagaimanapun, ACS mengalami masalah genangan dalam pengkomputeran grid bersaiz sederhana dan besar. ACS berasaskan mekanisma eksploitasi dan penerokaan di mana eksploitasi adalah mencukupi tetapi berkurangan pada penerokaan. Penerokaan dalam ACS adalah berasaskan kepada pendekatan rawak tanpa sebarang strategi. Kajian ini mencadangkan empat algoritma hibrid di antara ACS, Genetic Algorithm (GA), dan Tabu Search (TS) untuk meningkatkan prestasi ACS. Algoritma tersebut adalah ACS(GA), ACS+GA, ACS(TS), dan ACS+TS. Algoritma hibrid yang dicadangkan ini akan meningkatkan ACS dari segi mekanisma penerokaan dan penghalusan penyelesaian dengan melaksanakan penghibridan tahap rendah dan tinggi algoritma ACS, GA, dan TS. Semua algoritma yang dicadangkan telah dinilai dan dibandingkan dengan dua belas metaheuristic algoritma dalam persekitaran pengkomputeran grid statik (masa jangkaan kepada model pengiraan) dan dinamik (corak taburan). Satu simulator yang dinamakan ExSim telah dibangunkan untuk meniru sifat statik dan dinamik pengkomputeran grid. Keputusan eksperimen menunjukkan algoritma yang dicadangkan mengatasi ACS dari segi nilai makespan terbaik. Prestasi ACS(GA), ACS+GA, ACS(TS) dan ACS+TS adalah lebih baik daripada ACS dengan masing-masing meningkat sebanyak 0.35%, 2.03%, 4.65% dan 6.99% untuk persekitaran statik. Untuk persekitaran dinamik, prestasi ACS(GA), ACS+GA, ACS+TS dan ACS(TS) adalah lebih baik daripada ACS iaitu masing-masing meningkat sebanyak 0.01%, 0.56%, 1.16%, dan 1.26%. Algoritma yang dicadangkan boleh digunakan untuk penjadualan tugas dalam pengkomputeran grid dengan prestasi yang lebih baik dari segi makespan.

Kata Kunci: Algoritma metaheuristik, Ant colony system, Genetic algorithm, Tabu search, Penjadualan kerja dalam pengkomputeran grid.

Abstract

Grid computing is a distributed system with heterogeneous infrastructures. Resource management system (RMS) is one of the most important components which has great influence on the grid computing performance. The main part of RMS is the scheduler algorithm which has the responsibility to map submitted tasks to available resources. The complexity of scheduling problem is considered as a nondeterministic polynomial complete (NP-complete) problem and therefore, an intelligent algorithm is required to achieve better scheduling solution. One of the prominent intelligent algorithms is ant colony system (ACS) which is implemented widely to solve various types of scheduling problems. However, ACS suffers from stagnation problem in medium and large size grid computing system. ACS is based on exploitation and exploration mechanisms where the exploitation is sufficient but the exploration has a deficiency. The exploration in ACS is based on a random approach without any strategy. This study proposed four hybrid algorithms between ACS, Genetic Algorithm (GA), and Tabu Search (TS) algorithms to enhance the ACS performance. The algorithms are ACS(GA), ACS+GA, ACS(TS), and ACS+TS. These proposed hybrid algorithms will enhance ACS in terms of exploration mechanism and solution refinement by implementing low and high levels hybridization of ACS, GA, and TS algorithms. The proposed algorithms were evaluated against twelve metaheuristic algorithms in static (expected time to compute model) and dynamic (distribution pattern) grid computing environments. A simulator called ExSim was developed to mimic the static and dynamic nature of the grid computing. Experimental results show that the proposed algorithms outperform ACS in terms of best makespan values. Performance of ACS(GA), ACS+GA, ACS(TS), and ACS+TS are better than ACS by 0.35%, 2.03%, 4.65% and 6.99% respectively for static environment. For dynamic environment, performance of ACS(GA), ACS+GA, ACS+TS, and ACS(TS) are better than ACS by 0.01%, 0.56%, 1.16%, and 1.26% respectively. The proposed algorithms can be used to schedule tasks in grid computing with better performance in terms of makespan.

Keywords: Metaheuristic algorithms, Ant colony system, Genetic algorithm, Tabu search, Job scheduling in grid computing.

Acknowledgement

Each part of this study is guided, inspired, and supported by many people. The most important support and guidance were from my research supervisor Prof. Dr. Hjh. Ku Ruhana Ku Mahamud. Thank you very much for your great help and support. It is an honour for me to do a research under your supervision.

I would like to thank all the academic and technical staff in Utara Universiti Malaysia for their help in the study process and providing all the excellent facilities.

Furthermore, I would like to thank all the members of my family for their unconditional support. My goal would not be achieved without them.

Finally, I would like to thank all my friends for their support.

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
ACS	Ant Colony System
ACS(GA)	Low level hybridization between ACS and GA algorithms
ACS(TS)	Low level hybridization between ACS and TS algorithms
ACS+GA	High level hybridization between ACS and GA algorithms
ACS+TS	High level hybridization between ACS and TS algorithms
AS	Ant System
AS(GA)	Low level hybridization between AS and GA algorithms
AS(TS)	Low level hybridization between AS and TS algorithms
AS+GA	High level hybridization between AS and GA algorithms
AS+TS	High level hybridization between AS and TS algorithms
AS_{rank}	Rank-Based Ant System
BABC	Binary Artificial Bee Colony
BACO	Balanced Ant Colony Optimization
BFO	Bacterial Foraging Optimization
BWAS	Best-Worst Ant System
cMA	Cellular Memetic Algorithm
CV	Coefficient of Variation
DE	Differential Evolution
DETC	Dynamic Expected Time to Compute
EA	Evolutionary Algorithms

EAS	Elitist strategy for Ant System
EBABC1	Efficient Binary Artificial Bee Colony
EBABC2	Efficient Binary Artificial Bee Colony with flexible ranking
ET	Execution Time
ETC	Expected Time to Compute
FANT	Fast Ant System
FCFS	First Come First Served
FPLTF	Fastest Processor to Largest Task First
GA	Genetic Algorithm
GA(TS)	Low level hybridization between GA and TS algorithms
GA+TS	High level hybridization between GA and TS algorithms
GBF	Genetic Bacterial Foraging
GSA	Genetic and Simulated Annealing
GTSP	Generalized Traveling Salesman Problem
HACO	Hybrid Ant Colony Optimization
HC	Hill Climbing
HCACO	Hybrid Converse Ant Colony Optimization
HGS	Hierarchic Genetic Strategy
HPDSs	High Performance Distributed Systems
IACO	Improved Ant Algorithm
JSP	Job Scheduling Problem
KPB	K-Parents Best
LJFR-SJFR	Longest Job to Fastest Resource-Shortest Job to Fastest Resource
LM	Local Move
LMCTS	Local Minimum Completion Time Swap

MA	Memetic Algorithms
MA+TS	High level hybridization between Memetic and Tabu Search
MACO	Multiple Ant Colonies Optimization
MCT	Minimum Completion Time
MDS	Metacomputing Directory Service
MET	Minimum Execution Time
MI	Million Instructions
MIPS	Million Instructions Per Second
MMAS	Max-Min Ant System
MTEDD	Minimum Time Earliest Due Date
MTERD	Minimum Time Earliest Release Date
OLB	Optimization Load Balancing
PGA1	Player's Genetic Algorithm
PGA2	Parallel Genetic Algorithm
PMCT	Player's Minimum Completion Time
PSO	Particle Swarm Optimization
PSO-GELS	Particle Swarm Optimization and Gravitational Emulation Local Search
QoS	Quality of Service
RGA	Risky Genetic Algorithm
SA	Simulated Annealing
SGA	Struggle Genetic Algorithm
SLM	Steepest Local Move
SS	Scatter Search
SSGA	Steady-State Genetic Algorithm

SwA	Switching Algorithm
TS	Tabu Search
TSP	Traveling Salesman Problem

CHAPTER ONE

INTRODUCTION

The concept of grid systems goes back to 1969 when Leonard Kleinrock wrote, "We will probably see the spread of computer utilities, which, like present electric and telephone utilities, will service individual homes and offices across the country" (Wankar, 2008). From that time, many researchers presented various works and contributed in grid systems fields. The popularity of grid systems started by the late 1990s when Foster developed a grid system called Globus Toolkit (Foster & Kesselman, 1997; 2004).

Grid systems evolves from existing technology such as distributed computing, web service, and Internet (Magoules, Pan, Tan, & Kumar, 2009). According to Xhafa and Abraham (2010), grid computing is defined as "Geographically distributed computers, linked through the internet in a Grid-like manner, which are used to create virtual supercomputers of vast amount of computing capacity able to solve complex problem from e-Science in less time than known before".

Magoules, Nguyen, and Yu (2009) presented an extensive definition for grid systems as "A hardware and software infrastructure that provides transparent, dependable, pervasive and consistent access to large-scale distributed resources owned and shared by multiple administrative organizations in order to deliver support for a wide range of applications with the desired qualities of service. These applications can perform either: high throughput computing, on-demand computing, data intensive computing, or collaborative computing".

From previous definitions, it can be concluded that grid computing is a collection of geographically distributed and heterogeneous resources. They are connected like a grid using internet technology to form a virtual supercomputer that has the capacity to solve very complex problems. Grid can be used by different fields such as science, commerce, and education.

Grid systems could be distributed geographically through different organizations using different platforms (Kolodziej, 2012). Two services offered by grid systems are computing-intensive services and data-intensive services (Xhafa & Abraham, 2008b). In computing services, the grid process tasks are not possible or very difficult to be processed in traditional computer resource, while data storage services provide a storage which is available through many mirrors and servers. Grid computing provides powerful computation resources for complex tasks such as scientific research, stock markets, and business requirements for organizations. However, according to Xhafa and Abraham (2008), grid computing is still in the developmental stage, and there are many challenges to be addressed in the future.

Grid systems are classified as a modern High Performance Distributed Systems (HPDSs) along with the clusters and cloud systems (Kolodziej, 2012). However, there are crucial characteristics which are different between them such as scale, network type, administrative domain, and resources structure (AL-Fawair, 2009; Montes, Sanchez, & Perez, 2012). Grid systems are extended to many other types of grids such as enterprise grid, sensor grid, campus grid, global grid, pc grid, and utility grid (Babafemi, Sanjay, & Adigun, 2013; Conejero, Caminero, Carrion, & Tomas, 2014; Fulop, 2008; Kolodziej, 2012).

Grid computing requirements, usage, and definitions have changed with time. Berman, Fox, and Hey (2003) categorized the evolution of grid into three different generations. The first generation is called meta-computing environment, namely FAFNER and I-WAY projects. In the second generation, the grid technology is developed such as grid resource management, resource brokers and schedulers, grid portal, and complete integrated systems. Some projects developed at this era are Globus, Legion, and UNICORE. The third generation represents the integration between grid computing and web services technologies such as OGSI and WSRF.

According to Magoules, Pan, et al. (2009), grid architecture consists of four layers. The first layer is the user application level (APIs). The second layer is the middleware layer which includes management software and packages. The third layer deals with resources available to the grid such as data storage, processing capabilities and other application-specific hardware. The fourth layer deals with network components such as routers, switches, and the protocols used for communication between sources in the grid. In addition, the main characteristics of grid systems are distributed, non-dedicated, service-oriented heterogeneous, and non-centralized (Montes et al., 2012).

Magoules, Pan, et al. (2009) classified the usage of grid applications into five major groups: The first group is distributed computing which is the grid computing application to solve problems that cannot be solved on a single system, such as simulation of complex physical process, which needs many resources like CPU and memory. The second group is called high-throughput computing where the grid utilizes the unused processor cycles in order to perform independent tasks. Using this method, a complex task can be divided into multiple tasks and the grid will schedule and manage the process. Problems such as bio-statistical, molecular simulations of

liquid crystal and Monte Carlo simulations are very suited for high-throughput computing. The third group is on-demand computing. For this type, some resources cannot be cost-effectively or not locally located. Therefore, grid can provide an access to such a resource. Yet, there are many issues to be addressed, such as security, scheduling, code management, configuration, fault tolerance, and payments process. A typical example of an application requiring on-demand computing is the use of dynamically acquired supercomputer to perform cloud detection algorithm.

The fourth group is data intensive computing, which is a grid that can be used to manage data from distributed data repositories, digital library and database. A field such as High Energy Physics (HEP) is an example of application that requires data intensive computing support. The fifth group is called collaborative computing, whereby some applications require strict real time capabilities and different types of interactions that can take place. A typical example of such application that could use a collaborative computing is multi-conferencing.

One of the most important components in grid computing is resource management system (Hussain et al., 2013). Resource management system has the responsibility to map the submitted tasks to the available resources (Sim, 2009). Resource management system is implemented with scheduling algorithm to map tasks to the resources in an efficient way (Espling, 2013; Ma, Yan, Liu, Guan, & Lee, 2011). In grid computing, job scheduling algorithm is the main issue for grid computing performance (Kolodziej, 2012; Kousalya & Balasubramanie, 2009; Mathiyalagan, Suriya, & Sivanandam, 2010; Visalakshi & Sivanandam, 2009). Scheduling can be done in a simple way by assigning the incoming task to the available resource. However, by using a more advanced and sophisticated scheduler algorithm, the grid

can obtain better computing performance (Kolodziej, 2012). The scheduler should take into account many aspects, such as dynamic tasks environment, joining and dropping of resources from grid and evaluating the current load of the resources. The scheduler can be in hierarchical or distributed organization to deal with large scale of the grid.

In order to solve the grid computing scheduling problem, it is very important to define the problem first. The scheduling problem is defined as an NP-complete problem (Maheshbhai, 2011; Mao, 2011; Wei, Zhang, Li, & Li, 2012). The NP-complete problem needs heuristic or metaheuristics algorithms which have the ability to solve combinatorial optimization problems in a reasonable time (Aleti, 2012). Metaheuristics algorithms, such as Simulated Annealing (SA) algorithm, are implemented for job scheduling in grid computing (Cai, Ning, Li, & Zhong, 2007; Guo & Wang, 2010). However, SA needs a long running time to reach a good quality solution which is very restricted in grid computing environment (Xhafa, Barolli, & Durresi, 2007a). Genetic Algorithm (GA) is also implemented successfully in grid computing scheduling problems (Carretero, Xhafa, & Abraham, 2007). GA is able to find a good solution in consistent and semi-consistent environments. However, in inconsistent environment, GA suffers from premature convergence (Kolodziej, Xhafa, & Kolanko, 2009). Another important metaheuristics approach is Tabu Search (TS) algorithm which is implemented for job scheduling on computational grid system (Xhafa, Carretero, Dorronsoro, & Alba, 2009). TS algorithm has the benefit of systematic search which makes the algorithm avoid random solution. However, TS also suffers from stagnation due to the use of local neighbourhood search technique. One more important family of metaheuristics algorithms are ACO algorithms which are implemented widely on scheduling problems (Bandieramonte, Stefano, & Morana,

2008; Chang, Chang, & Lin, 2009; Cyril Daisy Christina & Miriam, 2012; Kant, Sharma, Agarwal, & Chandra, 2010; Zhu, Zhao, & He, 2010a).

ACO algorithms which are inspired by biological ants present many solutions for different types of NP-complete problems (Dorigo & Stutzle, 2004).

Biological ants have the ability to discover the shortest route from the nest to the source of food (Dorigo & Stutzle, 2004). Although they do not have an advanced vision system (Camazine et al., 2003), they have the ability to communicate with the environment. Ants use a chemical substance called "*pheromone*" to communicate with the environment and between each other (Dorigo & Gambardella, 1997a). Pheromone substance has evaporation property which is a powerful mechanism to update the route information. While an ant moves looking for food, it deposits a pheromone along the path. The following ant will more likely select the route with richer pheromone. This mechanism will make the ant choose the shortest path.

There are several enhanced ACO algorithms implemented in grid computing scheduling problems such as balanced job scheduling using Ant Colony Optimization (BACO) for grid environment by Chang, Chang, and Lin (2007). A study proposed by Chang, Chang, and Lin (2009) implemented ant algorithm for balanced job scheduling in computational grid. Kousalya and Balasubramanie (2009) presented a study on improving ant colony optimization algorithm with local search for job scheduling in computational grid systems. Liusuqin, Shuojun, Menglingfen, and Lixingsheng (2009) published a study to improve ant colony optimization for Job Scheduling Problem (JSP). A multiple ant colony model called "Cooperative multi-ant Colony Pseudo-parallel Optimization Algorithm" was proposed by Liu, Song, and

Dai (2010). Another study regarding ACO algorithm for job scheduling on computation grid was proposed by Kant, Sharma, Agarwal, and Chandra (2010). A research on task scheduling with load balancing using Multiple Ant Colonies Optimization (MACO) in grid computing was conducted by Bai et al. (2010). Enhanced ant colony algorithm for job scheduling in computational grid was proposed by Maruthanayagam and UmaRani (2010). Mou (2011) presented a new approach using double Pheromones technique for ant colony system. An improved ACO algorithm for job scheduling in computational grid systems proposed by MadadyarAdeh and Bagherzadeh (2011). Kokilavani and Amalarethinam (2013) published a study on implementing ant colony optimization based load sharing for job scheduling in computational grid systems.

The first version of ACO known as ant system algorithm was presented by Dorigo, Maniezzo, and Colorni (1991). Ant system is utilized to solve job scheduling on grid computing (Kousalya & Balasubramanie, 2008). Another version of ACO is elitist ant system algorithm which has better performance than ant system (Dorigo, Maniezzo, & Colorni, 1996). However, the performances of ant system and elitist ant system algorithms drop dramatically once the size of the problem instance increase (Dorigo & Stutzle, 2004). Another important version of ACO algorithms is Ant Colony System (ACS) presented by Dorigo and Gambardella (1997b). ACS algorithm mimics the behaviour of real ant colony to solve optimization problems such as Traveling Salesman Problem (TSP) and network routing. ACS algorithm is considered as one of the best algorithms for solving NP-complete problems (Gendreau & Potvin, 2010). Therefore, this study selected ACS as the main algorithm to hybridize with GA and TS algorithms.

In ACS algorithm, ants apply exploitation and exploration mechanism when they select the next node to move. In addition, ACS applies local pheromone update and global pheromone update to direct the search for the next iteration. The global update is calculated based on the quality of the best solution so far, while the local update applies the evaporation concept. In ACS algorithm, the exploration mechanism is based on a stochastic process. Such a random process will affect the whole solution if the ant selects a bad choice in any cycle of the solution construction process. Therefore, a more deterministic and systematic exploration mechanism is required to enhance ACS algorithm performance.

Hybridizing ACS algorithm with local search approaches such as genetic algorithm or tabu search algorithms will enhance the solutions produced by ACS. In spite of several enhanced ACS algorithms used for job scheduling problem, more studies are required to enhance the algorithm performance.

1.1 Problem Statement

In grid computing systems, many criteria depend on scheduling algorithm efficiency such as grid performance, utilization, Quality of Service (QoS), and load balancing (Nithya & Shanmugam, 2011; Zhu & Wei, 2010).

Xhafa & Abraham (2010) stated that "Rather than a problem, scheduling in grid systems is a family of problems. This is due to the many parameters that intervene scheduling as well as due to the different needs of grid-enabled applications".

In scheduling algorithm, there are many factors and parameters that should be taken into account, such as job size, resource capacity, network speed, current load, and

expected time to complete (AL-Fawair, 2009; Bai et al., 2010). In addition, the dynamic and heterogeneous nature of grid environment makes the scheduling process more critical such as joining and dropping resources to the gird (Maheshbhai, 2011; Malarvizhi & Uthariaraj, 2009; Qureshi et al., 2014). If the scheduling algorithm is not efficient, the grid system user will experience a delay in response time, especially when the number of tasks is increased (Ku-Mahamud & Nasir, 2010). Therefore, scheduling algorithm is a very important part in grid computing systems and it needs to be improved to answer the dynamic requirements. Hence, a sufficient algorithm for dynamic grid scheduling problem is demanded to enhance the grid computing system performance (Ku-Mahamud & Nasir, 2010; Maruthanayagam & UmaRani, 2010).

The traditional ACS is not efficient in large-scale computation problems due to the stagnation nature of pheromone in ACS (Mathiyalagan et al., 2010). ACS suffers from deficiency in the exploration mechanism. Experiments such as those in Dorigo and Stutzle (2004) used 0.1 for exploration and 0.9 for exploitation which indicate that the exploration mechanism is not sufficient, while the exploitation mechanism is efficient. However, because of the high ratio in exploitation, ACS algorithm has behaved like greedy algorithm more than metaheuristics algorithm. After observing the algorithm iteration step by step, it is found that the exploration is a stochastic process and not guided. As the ACS algorithm uses construction approach, any wrong selection for one or more node will affect the whole solution quality and that is what happens when the stochastic exploration has selected a wrong node. According to Glover and Laguna (1997), “bad strategic choice can yield more information than a good random choice”. Therefore, ACS algorithm needs to be enhanced in terms of exploration mechanism. In addition, ACS algorithm needs a mechanism to correct the construction phase after each cycle. Genetic algorithm and tabu search methods are

very good candidates to be hybridized with ACS as both algorithms complement ACS. In other words, ACS algorithm works based on constructive approach, while GA and TS algorithms work based on a local search which is suitable to be hybridized with ACS algorithm. Moreover, GA and TS algorithms could be hybridized as a low level as well as a high level with ACS.

The research questions answered in this study are as follows:

i. How to improve the job scheduling in static and dynamic grid computing.

ii. How to hybridize ant colony system algorithm with genetic algorithm to enhance the exploration and exploitation mechanisms?

iii. How to hybridize ant colony system algorithm with tabu search algorithm to enhance the exploration and exploitation mechanisms?

iv. How the hybridized algorithms will avoid the local optimum trap?

v. Can a simulator produce benchmark environment of the grid computing?

1.2 Objective of the Study

The main objective of the study is to develop a hybrid ant colony system algorithm in order to improve the job scheduling in static and dynamic grid computing environment.

Specific objectives of the study are:

i. To propose a hybrid low level algorithm to enhance the exploration mechanism in ACS algorithm.

ii. To propose a hybrid high level algorithm in refining the final solution found by ACS algorithm.

iii. To design and develop a simulator that can be used to simulate the static and dynamic grid environment.

iv. To evaluate the performance of the proposed hybrid ant colony system algorithm.

1.3 Significance of the Study

The developed hybrid algorithm can be considered as a new member in the family of ant colony optimization algorithms. The hybrid ACS algorithm is a new contribution to the body of knowledge in the area of swarm intelligence and job scheduling in grid computing.

The hybridization between ant colony system algorithm, genetic algorithm, and tabu search algorithm to enhance the exploration part inside ACS guided the ants' exploration in a better way. This hybridized algorithm concept can be used to solve other optimization problems with minimal customization. In addition, this study has implemented, compared and analysed sixteen algorithms for job scheduling in computational grid. The analyses provide a deep understanding regarding the behaviour of these algorithms.

The developed gird computing environment simulator has the ability to generate static and dynamic benchmark problems which are very useful and highly demanded in the field of job scheduling in grid computing. The simulator can be extended easily for other scheduling algorithms.

1.4 Scope and Assumption of the Study

This study has focused on a single ant colony system algorithm which is a variant of the ant colony optimization algorithms. The algorithm enhancement focused on low and high level hybridizations between ACS and local search algorithm to improve the exploration mechanism and enhance the final solution.

The implemented algorithm is applied on static and dynamic environments using batch mode scheduling and independent task approach. For static environment, the experiments were conducted using benchmarks presented by Ali, Siegel, Maheswaran, Hensgen, and Ali (2000). This benchmark is frequently used because of its effectiveness in simulating job scheduling problems in grid systems (Xhafa, Alba, Dorronsoro, Duran, & Abraham, 2008). For dynamic environment, a simulator was developed to mimic the real grid computing environment using the same pattern of distribution (Feitelson, 2013).

This study has investigated GA and TS algorithms to enhance the exploration mechanism and refine the final solution in ACS algorithms. In addition, this study used three performance metrics to measure the grid computing performance, namely makespan, flowtime, and utilization.

1.5 Thesis Organization

This thesis is organized as follows: Chapter 2 presents a review on various scheduling algorithms in computational grid, hybrid approaches in metaheuristics algorithms, grid simulator, and conceptual framework. The research framework, methods, and techniques are discussed in Chapter 3. Chapter 4 presents the simulator development steps with verification and validation. Details regarding the problem formulation,

parameters, performance criteria, and static experiments are provided in Chapter 5. Chapter 6 presents the experiments on dynamic environment. Chapter 7 discusses the contributions, limitations, and concludes the study with suggestion for future research.

CHAPTER TWO

LITERATURE REVIEW

This chapter presents the review of previous studies that have been done in the areas of job scheduling in computational grid systems, swarm intelligence, scheduling algorithms based on hybrid approaches, and grid computing simulator.

Job scheduling is the heart of grid computing processes. One of the main issues in job scheduling for grid computing is the dynamic environment. Many studies try to solve this problem using different techniques and algorithms. This chapter reviews some of these studies and points out how they evolved and developed. In addition, their limits and gaps will be highlighted.

This chapter organized as follows: Section 2.1 discusses several algorithms implemented in job scheduling on computational grid. Hybrid approaches for job scheduling in grid computing are presented in Section 2.2. Studies on grid computing simulator are discussed in Section 2.3. Section 3.4 presents the conceptual framework. Finally, the chapter summary is presented in Section 2.5.

2.1 Job Scheduling Algorithms in Computational Grid System

Grid computing system can be categorized into two types, namely static and dynamic environments (Kolodziej, 2012). In static environment, resources are assumed to be available at all time. In addition, the resource capacity, number of tasks and load are also fixed. For dynamic environment, resources may join and leave the grid at any time. In addition, the resource capacity and load changes dynamically. Other characteristics of the dynamic environment are variation of network speed and

availability. The performance of the static and dynamic grid computing depends upon the efficiency of scheduler algorithm. The scheduler is considered as one of the most important parts of resource management system in grid computing system. The scheduler is implemented with heuristic or metaheuristics algorithms. The following reviews discuss several types of algorithms which are implemented successfully to solve job scheduling problems in computational grid.

2.1.1 Heuristic Algorithms

In general, a scheduling problem is considered as a complex decision making problem which needs to be solved optimally (Zapfel, Braune, & Bogl, 2010). This type of problem is known as optimization problem which is also categorized as NP-complete problem (Abraham, Grosan, & Ishibuchi, 2007; Talbi, 2013b). Due to the complexity of NP-complete problems, using exact approaches are not feasible (Framinan, Leisten, & García, 2014). Therefore, the optimal solution could be sacrificed for the sake of finding good solution in significantly reduced time using approximate algorithms (Blum & Roli, 2003). Heuristics and metaheuristics algorithms have been implemented to solve various optimization problems, such as routing, scheduling, and planning (Agnetis, Billaut, Gawiejnowicz, Pacciarelli, & Soukhal, 2014; Burke & Kendall, 2014). A simple approach could be used for job scheduling in grid computing based on greedy approach (Boussaid, Lepagnot, & Siarry, 2013; Ma, Lu, Zhang, & Sun, 2012). These types of algorithms provide fast solution which is suitable for extremely small and time restricted grid computing systems (Xhafa & Abraham, 2009).

A study on batch mode scheduling for job scheduling in computational grid systems published by Xhafa, Barolli, and Durresi (2007b). The study aims to optimize four

criteria, namely makespan, flowtime, resource utilization, and matching proximity. The algorithms proposed in their study are: Min-Min, Max-Min, Sufferage, Relative Cost, and Longest Job to Fastest Resource-Shortest Job to Fastest Resource algorithms. These algorithms evaluated using static benchmark problems based on expected time to compute model defined by Braun et al. (2001). The experiment results show that Min-Min algorithm outperforms other algorithms in term of makespan and flowtime. In terms of average resource utilization, Max-Min algorithm achieved the best results among all other algorithms. However, the results show that Min-Min algorithm achieved the worst performance in terms of average resource utilization. For matching proximity criterion, Min-Min algorithm achieved the best performance closely followed by Relative Cost algorithm. The authors recommended that the proposed algorithms are useful to be utilized to generate an initial solution for other heuristics algorithms.

An immediate mode scheduling of independent job in grid computing proposed by Xhafa, Barolli, and Durresi (2007c). Several heuristic algorithms were implemented and four criteria were used to measure the grid system performance, namely makespan, flowtime, resource utilization, and matching proximity. The study implemented and evaluated five algorithms, which are, Optimization Load Balancing (OLB), Minimum Completion Time (MCT), Minimum Execution Time (MET), Switching Algorithm (SwA), and K-Parents Best (KPB). These methods were tested using benchmark problems based on the expected time to compute model proposed by Braun et al. (2001). The experiments show that MCT algorithm performs the best in terms of makespan. Regarding flowtime criterion, SwA algorithm performs good, followed by MCT. In terms of resource utilization, OLB algorithm outperforms other algorithms. MET algorithm was able to achieve the best matching proximity values

with perfect results. It is clear from the results that the objectives of optimizing job scheduling in computational grid system have some contradictions. Therefore, for very heterogeneous systems, not all criteria are considered as a good indicator for grid system performance.

A study on job scheduling algorithm in computational grid environment conducted by Malarvizhi and Uthariaraj (2009) for grid job scheduling using minimum Time To Release algorithm (TTR). They proposed architecture for grid scheduling; some of the main architecture components are dispatcher, grid scheduler and load balancer. The idea behind their approach is to predict the performance of each resource by estimating the total time to release, and then map with each resource. Based on TTR, each combination of job and resource are stored in an increasing order of TTR to assign to a resource. They also calculate other parameters, such as transfer input time and transfer output time which makes the module more realistic and accurate for real life applications. The experiments were conducted using GridSim simulator and the environment consists of scheduler, five users with different time requirements and rates of task creating, and 30 nodes with different computer power. They compared TTR algorithm with first come first serve algorithm and Min-Min algorithm. The results show that the proposed algorithm performs better than the others in terms of computation time. However, according to Xhafa and Abraham (2010), metaheuristics algorithms such as ant colony optimization and genetic algorithm are much better than the other types of algorithm.

A study presented by Izakian, Abraham, and Snsel (2009) proposed a heuristic method called Min-Max to generate a solution for job scheduling in grid environment. Min-Max algorithm could be used to produce a good initial solution for other

metaheuristics algorithms. In addition, Min-Max algorithm could be used in real-world situations where applying metaheuristics algorithm is not applicable. The study focused on static environment which is based on expected time to compute model presented by Braun et al. (2001). The proposed algorithm was compared with five heuristic algorithms, namely Work Queue, Max-Min, LJFR-SJFR, Suffrage, and Min-Min. The experiment results show that the proposed algorithm can minimize the makespan and flowtime better than other algorithms. The study also conducted experiment using the proposed algorithm to generate initial solution for simulated annealing algorithm. The results show that simulated annealing algorithm could not improve the solution that generated using the proposed algorithm. In addition, reducing the makespan leads to increasing in the flowtime and vice versa. However, such contradiction behaviour is common in the problem of job scheduling in grid computing.

Bardsiri and Hashemi (2012) published a study on comparing seven static mapping heuristics algorithms for job scheduling on computational grid. The study compared seven popular heuristics algorithm, namely Opportunistic Load Balancing (OLB), Minimum Completion Time (MCT), Min-Min, Max-Min, Sufferage, Maxstd, and Longest Job to Fastest Resource-Shortest Job to Fastest Resource (LJFR-SJFR) algorithms. The proposed approaches were tested using the benchmark problems based on expected time to compute model developed by Braun et al. (2001). The study aims to optimize makespan, resource utilization, and matching proximity criteria for independent job scheduling in static grid computing environment. The experiment results show that Min-Min and Sufferage algorithms achieved good performance compared with other algorithms in terms of makespan. In terms of resource utilization, all algorithms have similar performance with little favour to Max-

Min algorithm. For matching proximity criterion, Min-Min algorithm outperforms the other algorithms followed by Sufferage algorithm. However, the experiments were conducted using static environment which is not enough to explore the algorithm behaviour. Nevertheless, these algorithms are useful to generate an initial solution for other metaheuristics algorithms such as tabu search and genetic algorithm.

2.1.2 Evolutionary Algorithms

In spite of the fast solutions produced by schedulers based on greedy approach, the quality of these solutions will drop down dramatically once the grid size increase (Anousha, Anousha, & Ahmadi, 2014; Braun et al., 2001). In order to overcome the obstacle of size increasing, metaheuristics algorithms emerged as a practical solution for job scheduling with reasonable time and resources (Qureshi et al., 2014). One important category of metaheuristics algorithms is Evolutionary Algorithms (EA) (Yu & Gen, 2010). The process in EA algorithms is similar to natural process in living organic such as crossover, mutation, and selection (Stoean & Stoean, 2014). Several algorithms are developed under the EA category such as genetic algorithm (Holland, 1992), evolution strategies (Schwefel, 1995), evolutionary programming (Fogel, Owens, & Walsh, 1996), and genetic programming (Koza et al., 2003). Evolutionary algorithms have been implemented successfully for job scheduling in computational grid systems with good results. Some of the EA works are discussed such as memetic algorithms, differential evolution algorithms, and genetic algorithms.

I. Memetic Algorithms

Memetic Algorithms (MA) is an optimization technique which combines different search methods such as population based search and local search algorithm (Moscato

& Cotta, 2010). The idea behind MA is that individuals are not simply doing crossover and mutation, but also doing some enhancement to their solution elements. This enhancement is accomplished by incorporating heuristic, approximation algorithms, or local search technique (Gendreau & Potvin, 2010). Due to the combination of different techniques, MA is sometimes called hybrid evolutionary algorithms (Davis, 1991). Memetic algorithms are applied to job scheduling in grid computing by several studies.

Cellular Memetic Algorithm (cMA) proposed to optimize the makespan and flowtime for job scheduling in grid computing by Xhafa, Alba, and Dorronsoro (2007). As a component of cMA algorithm, the study implemented three types of local search algorithms: Local Move (LM), Steepest Local Move (SLM), and Local Minimum Completion Time Swap (LMCTS). The experiments conducted using static environment based on expected time to compute model presented by Braun et al. (2001). In addition, the proposed algorithm were evaluated with three other versions of GA as presented in Braun et al. (2001), Carretero and Xhafa (2006), and Xhafa (2006). The first experiment shows that the local search algorithm LMCTS is the best among the three considered algorithms. Experiment on makespan criterion shows that the proposed cMA performs better than other algorithms in some instances. However, the experiments were not organized properly for example, the flowtime results were obtained only from comparing cMA with LJFR-SJFR and GA from Xhafa (2006) and nothing was reported about the other two GA algorithms. Moreover, the study focused only on static grid computing environment. Therefore, more study and experiments using dynamic grid computing environment are required to explore the algorithm behaviour.

Xhafa, Alba, Dorronsoro, Duran, and Abraham (2008) published a study using Cellular Memetic Algorithm (cMA) for job scheduling in grid system. The study aimed to optimize makespan and flowtime simultaneously as a bi-objective. The authors implemented two different local moves: Local Minimum Completion Time Swap (LMCTS), and Local Tabu Hop (LTH) based on Tabu Search (Glover & Laguna, 1997). Therefore, the study proposed two different cMA algorithms, namely: cMA+LMCTS and cMA+LTH. The proposed approaches were evaluated with static and dynamic grid computing environments. The static environment benchmark problems were generated using expected time to compute model presented in Braun et al. (2001). On the other hand, the dynamic environment benchmark problems generated using a simulator developed by Xhafa, Carretero, Barolli, and Durresi (2007). The proposed algorithms were evaluated first with TS as presented by Xhafa, Carretero, Alba, and Dorronsoro (2008) in order to select the best move using static scenario. The first experiment shows that cMA+LTH algorithm was able to achieve good makespan reduction on five instances while cMA+LMCTS algorithm could not achieve any good results. On the other hand, TS achieved the best results on seven instances. Nevertheless, cMA+LMCTS performs better than cMA+LTH with flowtime reduction. The second experiment compared cMA+LTH performance with three other versions of GA approaches as presented by Braun et al. (2001), Carretero and Xhafa (2006), and Xhafa (2007). The empirical results show that cMA+LTH outperforms other algorithms in static and dynamic environments. It is clear from the results that some reduction in makespan value will lead to increase in flowtime values. Although the study was conducted using static and dynamic grid computing environments, the comparison is not sufficient, for example, the authors did not

compare with TS in dynamic scenario. However, the concept of utilizing local search algorithm inside other metaheuristics algorithm is quite useful.

Zhong, Long, Zhang, and Song (2011) published a study on efficient memetic algorithm for scheduling job in grid computing. The authors incorporated hill-climbing and tabu search algorithms as solution enhancement mechanisms. The experiments were conducted using benchmark problems based on expected time to compute model as introduced by Braun et al. (2001) and the fitness function to minimize makespan and flowtime values as proposed in Xhafa, Alba, et al. (2008). The proposed algorithms evaluated against genetic algorithm described in Braun et al. (2001). The experiment results show that MA with hill-climbing and tabu search algorithms outperforms genetic algorithm for consistent and semi-consistent grid scenarios. Comparing between MA-Tabu search and MA-hill-climbing, MA-tabu search showed faster performance. However, the flowtime results are not reported which is supposed to be part of the optimization function. In addition, an experiment conducted only on static environment is not enough to conclude the algorithm behaviour on dynamic scenario. Moreover, the proposed algorithm did not compare with other recent metaheuristics algorithms such as artificial bee colony and ant colony optimization. Nevertheless, the idea of using tabu search algorithm as a mechanism to enhance the individual solution could be integrated with other metaheuristics algorithms such as ant colony optimization.

A comparison study on the performance of genetic algorithm, memetic algorithm, cellular memetic algorithm, and hybrid algorithms with tabu search has been proposed by Xhafa, Koodziej, Duran, Bogdanski, and Barolli (2011). The study illustrated the advantages and limitations of different population based methods for job scheduling

in computational grid systems. In addition, the study tried to investigate the benefits of hybridizing population algorithm with local search algorithms such as tabu search. The authors considered a bi-objective scheduling problem in grid computing to measure the scheduling effectiveness, namely makespan and flowtime which optimized simultaneously. The study considered the tasks to be processed in a batch mode as described in Xhafa and Abraham (2010). In addition, the problem was formulized based on expected time to computed matrix model as proposed by Ali et al. (2000b). The experimental analysis was performed using HyperSim-G simulator as developed by Xhafa et al. (2007). These experiments were conducted on static and dynamic instances. The experiment results for static instances show that memetic algorithm achieved the best makespan value for large instances, while cellular memetic algorithm hybridized with tabu search achieved the best results for small instances. The dynamic experiments results show that the hybrid cellular memetic algorithm with tabu search outperforms the other algorithms for small, medium, and large instances, while memetic algorithm hybridized with tabu search achieved the best makespan values for very large instances. The study concluded that hybridizing memetic and cellular memetic algorithms with tabu search will enhance the algorithm performance. However, the study did not show the result of the flowtime values which is considered as a part of the bi-objective function. Nevertheless, the proposed comparison provides good foundation regarding the performance of memetic and cellular memetic algorithm when they hybridize with tabu search algorithm.

II. Differential Evolution Algorithm

Differential Evolution (DE) is an optimization algorithm developed by Kenneth Price in 1995 (Price, Storn, & Lampinen, 2005). DE is a population-based algorithm that

has the operators: crossover, mutation, and selection to evolve a population of candidate solutions toward an optimal solution. Differential evolution algorithm for job scheduling in heterogeneous distributed environment has been proposed by Kromer, Snasel, Platos, Abraham, and Izakian (2009). The study aimed to optimize the job scheduling in terms of makespan and flowtime with priority to makespan as suggested by Carretero et al. (2007). The proposed algorithm was implemented with classic version adopted from Price et al. (2005). The experiments were conducted using the expected time to compute model as proposed by Braun et al. (2001). In addition, the experiments compared the proposed algorithm with max-min, suffrage, min-min, and min-max algorithms. The experiment results show that the proposed algorithm did not achieve a good result when it started with random initial solution. However, with initial solution generated using the heuristic algorithm, the proposed algorithm outperforms all other algorithms in terms of makespan. Nevertheless, in terms of flowtime optimization, the proposed algorithm did not perform well.

Selvi and Manimegalai (2010) conducted a study on job scheduling for grid computing based on differential evolution algorithm. The objective of the study is to minimize the makespan value as an optimization objective. The experiments were conducted using static benchmark problems used by Liu, Abraham, and Hassanien (2010). The proposed algorithm was implemented using MATLAB application. The performance of the implemented algorithm is compared with fuzzy discrete particle swarm optimization algorithm as proposed in H. Liu et al. (2010). The experiment results show that the proposed algorithm achieved good standard deviation and completion time. However, in terms of makespan values, the proposed approach did not achieve good results compared to other algorithms. In addition, the experiments

were conducted using static benchmark problems. Therefore, the proposed algorithm needs to be tested on dynamic environment in order to draw the final conclusion.

III. Genetic Algorithm

Genetic Algorithm (GA) is a well-known algorithm to solve various types of combinatorial optimization problems developed in 1975 by John Holland (Blum & Roli, 2003). Genetic algorithm is applied in various types of scheduling problems, such as manufacturing scheduling (Gen, Zhang, Lin, & Jo, 2014), scheduling of production and transport systems (Hartmann, Makuschewitz, Frazzon, & Scholz-Reiter, 2014), and scheduling workflow applications in cloud environment (Singh & Singh, 2014).

GA has three prime operators, namely crossover, mutation, and selection (Yang, 2014). However, In terms of mathematics, there are no explicit mathematical equations for general genetic algorithm (Yang, 2014). GA procedure's details, such as steps on how to generate a new generation from a population and how to process the operators are provided in many literatures (Reeves & Rowe, 2003; Sivanandam & Deepa, 2008). Figure 2.1 shows the pseudocode of basic genetic algorithm.

Procedure Genetic Algorithm

Step 1- $P \leftarrow$ initial population;

Step 2- Evaluate (P);

Step 3- **While** *termination criterion not satisfied* **do**

Step 4- $P' \leftarrow \text{Select}(P)$;

Step 5- $\text{Crossover}(P')$;

Step 6- $\text{Mutate}(P')$;

Step 7- $\text{Evaluate}(P')$;

Step 8- $P \leftarrow Replace(P' \cup P)$;

Step 9- **End**

End Procedure

Figure 2.1. Basic Genetic Algorithm (Zapfel et al., 2010)

In Figure 2.1, the first step is initializing the population (*P*) which is generated randomly or using some heuristic algorithm (Carretero & Xhafa, 2006). Figure 2.2 visualizing the GA population (Zapfel et al., 2010).

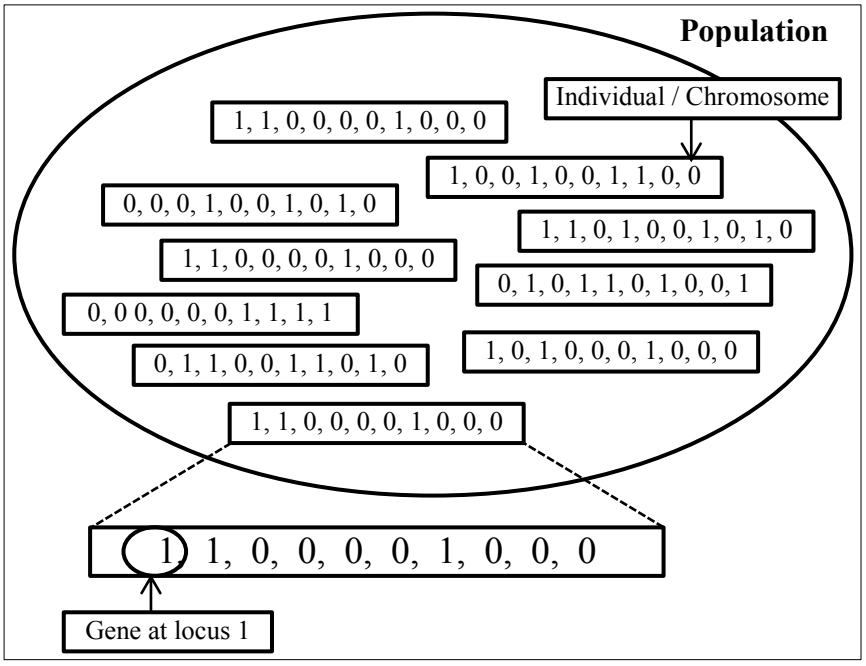

Figure 2.2. Visualization of GA population (Zapfel et al., 2010)

The second step in GA is the evaluation process. Evaluation is an operator to calculate the solution quality which is called fitness in terms of genetic algorithm. The solution fitness is required for selection and replacement operators. In other words, solutions with better fitness value are preferred in the selection process (Zapfel et al., 2010). In order to calculate the fitness value in job scheduling problem in grid computing, the following equation is used:

$$fitness = 1 / makespan \tag{2.1}$$

where $makespan$ is the completion time of the last task by the system (Xhafa & Abraham, 2010).

The third step in GA algorithm is the loop using while and termination condition. The execution could be stopped using one or more conditions, such as, specific number of iteration, specific time of execution, and maximum number of iteration without any enhancement in the solution quality (Zapfel et al., 2010).

The fourth step in GA algorithm is the selection process which is responsible to select part of the population for crossover and mutation operators. There several selection methods, such as Roulette-Wheel-Selection, Linear-Rank-Selection, and Tournament Selection (Zapfel et al., 2010).

The fifth step in GA is the recombination or crossover operators. Crossover operator is the process of combining genes from the selected solutions in order to produce a new solution. There are several types of crossover operator, such as one-point, two-point, N-point, and uniform crossover (Zapfel et al., 2010). Figure 2.3 shows examples of one-point, two-point, and uniform crossover.

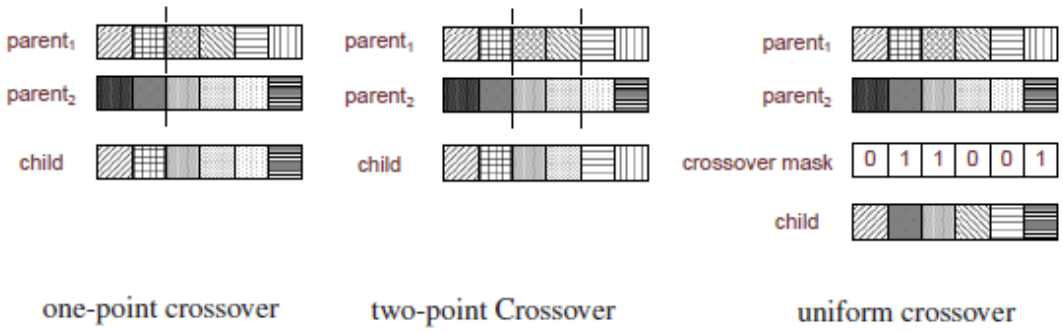

Figure 2.3. Examples of crossover operators (Zapfel et al., 2010)

The sixth step in GA algorithm is the mutation operator. Mutation is the process of perturbation of the solution with small probability. Mutation operator will help the algorithm to prevent from premature convergence. In binary solution representation, the mutation process could be done by changing the value from 1 to 0 or vice versa. This method is known as bit-flip mutation. In job scheduling problem for grid computing, mutation could be implemented using re-balance method (Xhafa, Duran, Abraham, & Dahal, 2008).

The seventh step in GA is the evaluation of the new solutions. This step is similar to step 2 using the new solutions instead of the whole population.

The last step in GA is the replace operator which replaces the new generated solution with other solutions from the population. There are several replacement methods, such as generational replacement and steady state replacement. In generational method, the new generated solutions supersede the old solutions. In steady state method, multiple solutions or only one solution is replaced. Figure 2.4 illustrates the process of genetic algorithm.

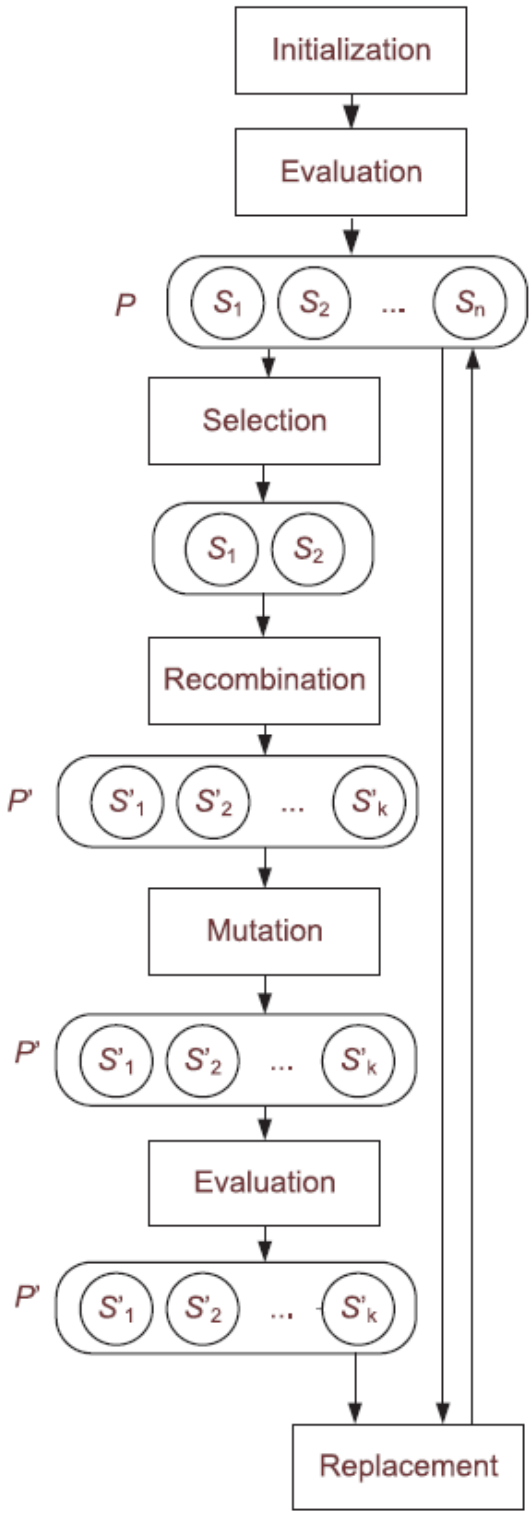

Figure 2.4. Process of Genetic Algorithm (Zapfel et al., 2010)

An experimental study regarding job scheduler based genetic algorithm for large grid environment has been proposed by Carretero and Xhafa (2006). The proposed algorithm aims to minimize the makespan and flowtime values of job scheduling on

grid computing. The study considered two versions of optimization. First is hierarchical structure where makespan is optimized first, then followed by flowtime. Second is simultaneous structure where both objectives are optimized simultaneously. The proposed algorithm was implemented using a skeleton as defined in Alba et al. (2002). The study conducted static and dynamic experiments. The static benchmark problems were generated using expected time to compute model from Braun et al. (2001), while dynamic benchmark problems were generated using dynamic simulator developed by Alba et al. (2002). For static experiments, the proposed algorithm compared with Min-Min, MCT, and GA algorithms from Braun et al. (2001). The results show that the proposed genetic algorithm with hierarchic optimization structure outperforms other algorithms in terms of makespan. However, flowtime results for static experiments are not reported in the study. For dynamic experiments, the results show that the proposed algorithm with hierarchical structure achieved the best makespan values for three instances, while simultaneous structure achieved the best results for one instance in terms of makespan. Flowtime results were not reported as well. The study provided the implementation details for genetic algorithm. However, the proposed algorithm was compared only with heuristic algorithms and other genetic algorithm implementation. Therefore, it is unknown how is the performance of the proposed algorithm compared with other metaheuristics algorithms such as ant colony optimization and artificial bee colony.

Job scheduling for grid computing based on genetic algorithm has been proposed by Carretero et al. (2007). The study aimed to minimize makespan and flowtime values either in a hierarchical mode with makespan as primary objective or in a simultaneous mode. In addition, two types of encoding schemes have been presented in the study with several operators implementation. The proposed algorithm was implemented

based on a generic skeleton as developed by Alba et al. (2006). The benchmark problems were generated using expected time to compute model presented in Braun et al. (2001) which consists of twelve static instances. The implemented algorithm was evaluated against genetic algorithm results as presented in Braun et al. (2001). The experiment results show that the proposed algorithm outperforms other algorithms in terms of makespan with simultaneous mode. However, flowtime values are not provided in Braun et al. (2001). The study compared the proposed algorithm in terms of flowtime values between hierarchical and simultaneous modes which show favour results to simultaneous approach. The authors also noticed that makespan and flowtime are contradictory objectives. Therefore, trying to optimize one objective may not suit the other objective, especially when the scheduling plan is close to optimal solution. In spite of the good results achieved by the proposed algorithm, more investigations are required especially under dynamic job scheduling environment in order to conclude the algorithm performance in all circumstances.

Another experimental study on genetic algorithm for job scheduling on grid computing has been proposed by Xhafa, Barolli, and Durresi (2007a). The authors presented two algorithms for scheduling independent jobs to grid resources based on two replacement mechanisms, namely Steady-State Genetic Algorithm (SSGA) and Struggle Genetic Algorithm (SGA). The study experiments were conducted using benchmark problems generated using expected time to compute model as presented in Braun et al. (2001). The proposed algorithm was compared with genetic algorithm developed by Braun et al. (2001). The experiment results show that SGA performs better than SSGA for moderate grid size problems in terms of makespan. However, for larger grid size problems, SGA did not achieve good results as SSGA. In terms of flowtime, SGA also achieved better performance than SSGA. The study

recommended implementing SGA for small and medium grid size and SSGA for large and very large grid size. However, the study only considered static grid computing environment which is not enough to observe the algorithm behaviour. Therefore, these two algorithms need to be tested on dynamic grid computing environment in order to conclude the algorithm robustness.

Xhafa, Duran, Abraham, and Dahal (2008) published a study on Straggle strategy in Genetic Algorithm (SGA) for job scheduling in computational grid. The study implemented hash function for computing the similarity of solutions in order to enhance the standard similarity measures. The aim of the proposed approach is to minimize the makespan value as an optimization objective. The proposed algorithm was evaluated with static benchmark problems generated using the expected time to compute model as presented in Braun et al. (2001). The evaluation is done between SGA and SGA with hash function. The experiment results show that using the hash function with SGA, it improves the algorithm performance in terms of makespan value for several types of instances. The idea of using hash function is useful in order to avoid evaluating a similar solution for each cycle in the algorithm process which is time consuming. This idea could be implemented with other metaheuristics algorithms such as tabu search and ant colony optimization. However, more studies and experiments are required to observe the algorithm behaviour in terms of flowtime and resource utilization. In addition, more experiments on job scheduling in dynamic grid computing environment will provide more information regarding the use of hash function.

A hierarchic genetic algorithm scheduler of jobs in grid computing environment has been published by Kolodziej, Xhafa, and Kolanko (2009). The authors proposed an

algorithm called Hierarchic Genetic Strategy (HGS) for job scheduling on grid computing systems. The study aimed to optimize the makespan and flowtime values simultaneously. In addition, the study objective is to investigate several variations of HGS operators and parameters to identify the best configuration for job scheduling problem. The experiments were conducted using static benchmark instances for job scheduling on grid computing which is based on the expected time to compute model as developed by Ali et al. (2000b). The proposed algorithm was compared with classic genetic algorithms as presented by Braun et al. (2001) and Carretero et al. (2007). The experiment results show that the proposed algorithm performs better than the other two algorithms in terms of makespan and flowtime as well. The study also revealed that rebalancing method is the best mutation operator for HGS. However, the experiments were conducted on static grid computing environment; therefore, more experiments are required to test the proposed algorithm on dynamic environment to examine the algorithm robustness.

A study conducted by Kumar, Kumar, and Kumar (2011) for job scheduling used genetic algorithm. In their study, they considered the network transmission time when making scheduling decisions. They argue that the scheduler who does not take into account the network load when making the scheduling decisions might not produce optimal scheduling. In their work, they implemented multi-objective genetic algorithm for job scheduling in grid computing using GridSim simulator as developed by Buyya and Murshed (2002). The algorithm focused on minimizing the jobs' finalization time and makespan by minimizing the jobs' data transfer time between data storage location and computing resource site over the network. The job size was presented in Million Instructions (MI) and the resource capacity was presented in Million Instructions Per Second (MIPS). To calculate the network load in the

scheduling algorithm, they used four arrays. The proposed algorithm was compared with non-network-aware scheduling algorithm for grid computing. The results show that the proposed scheduling algorithm performs better than the non-network-aware scheduling algorithm. However, the experiments scenario is very limited (using only 50 jobs) which did not give a clear robustness picture about the algorithm. Nevertheless, the idea of calculating the network transmission time is very important when scheduling jobs in grid systems and needs more investigation to concrete the concept.

Enhanced Genetic-based scheduling for grid computing has been developed by Kolodziej and Xhafa (2011). The authors presented an implementation of hierarchic genetic strategy for job scheduling in dynamic computational grid environment. HGS has the ability to search the solution space concurrently using various evolutionary processes. The study focused on bi-objective optimization, specifically, makespan and flowtime to be simultaneously optimized. The experiments were conducted under heterogeneous, large scale, and dynamic environments using grid simulator. The algorithm was tested with static and dynamic grid computing environments. The experiment with static environment is based on expected time to compute matrix model as presented in Ali et al. (2000b). For dynamic environment, the authors used a simulator developed by Xhafa and Carretero (2009). The proposed algorithm was compared with two other GA-based schedulers presented in Braun et al.(2001), and Carretero et al. (2007). The results show that the proposed algorithm outperforms the other GA-based scheduler algorithms. However, the proposed algorithm results were compared only with GA. Therefore, it is unknown how the proposed algorithm will perform against other metaheuristics algorithms.

2.1.3 Local Search

The basic idea of local search is that solution is successively modified by performing moves that change solution locally (Vob, 2001). The solutions that can be reached by moves are called neighbourhood solutions. Various techniques have been developed to search the solution's neighbourhood, such as random, iterated, greedy, variable neighbourhood search, and steepest descent algorithms (Aarts & Lenstra, 2003; Gendreau & Potvin, 2010; Zapfel et al., 2010). However, using metaheuristics algorithms for local search, such as simulated annealing and tabu search, showed promising performance in grid computing.

I. Simulated Annealing

Simulated Annealing (SA) is known as an optimization algorithm inspired by nature. SA mimics the physical process in the annealing of materials when a metal cools and freezes into a crystalline state (Yang, 2014). SA algorithm was developed in 1980s by Kirkpatrick, Gelatt, and Vecchi (1983). Simulated annealing algorithm has been implemented in almost every field of combinatorial optimization problems (Yang, 2014). The main advantages using SA is the ability to skip from local minima by controlling the threshold value for the maximum allowed decrease in solution quality (Vidal, 1993; Zapfel et al., 2010). SA algorithm has been proved to converge to the global optimality if enough time and randomness are given with very slow cooling (Aarts, Korst, & Michiels, 2014; Yang, 2014). SA algorithm has been implemented for job scheduling in computational grid environment successfully.

YarKhan and Dongarra (2002) conducted an experimental study on using Simulated Annealing (SA) algorithm for job scheduling in grid environment. The study used dynamic machine status and connectivity information from the Globus

Metacomputing Directory Service (MDS) defined in YarKhan and Dongarra (2002) and Network Weather System (NWS) proposed by Wolski, Spring, and Hayes (1999). The proposed algorithm was compared with Ad-hoc greedy approach presented in Ullman (1975). The experiments were conducted using real grid computing system provided from the University of Tennessee and University of Illinois. The experiment results show that the proposed simulated annealing algorithm outperforms Ad-hoc greedy algorithm in terms of estimated execution time. In spite of the practical experiments using real grid environment, the experiments were very limited in terms of benchmark problems size. In addition, the proposed approach needs to be compared with other metaheuristics algorithms in order to observe the algorithm performance. However, using simulated annealing algorithm will help to avoid the local minima and search near optimal solution.

A study on implementing simulated annealing algorithm for job scheduling in computational grid has been proposed by Fidanova (2006). The authors claimed that simulated annealing has two important features, which are: First, SA algorithm is proved to converge to the optimal solution. Second, AS algorithm is easy to be implemented to solve job scheduling problems in grid computing. The implemented scheduler utilizes the expected time to compute model which is developed by Maheswaran, Ali, Siegel, Hensgen, and Freund (1999). The objective of the study is to minimize the makespan value for job scheduling as an optimization function in batch mode. The proposed algorithm starts with generating initial solution using greedy heuristic approach. The performance of the proposed AS algorithm compared with online (queue) algorithm and ant colony optimization provided by Fidanova and Durchova (2006). The experiments were conducted using a scenario developed by the authors which consists of 5 resources and 20 jobs. In terms of makespan, the results

show that the proposed algorithm outperforms other algorithms followed by ant colony optimization algorithm. However, the experiment scenario is very small in terms of the number of resources and jobs. In addition, the study only focused on static grid computing environment. Therefore, designing and conducting a bigger scenario in static and dynamic environments will reveal the real performance of simulated annealing algorithm for job scheduling in grid computing.

Cai, Ning, Li, and Zhong (2007) published a study on using simulated annealing algorithm for independent task assignments in heterogeneous computing system. The authors proposed two neighbourhood structures to search the best neighbour of the current solution in the search space. The study aimed to minimize the makespan value as an optimization objective. The experiments were conducted using benchmark problems based on expected time to compute model defined in Braun et al. (1999). The proposed algorithm was compared with Min-Min algorithm and simulated annealing approach proposed by Braun et al. (1999). The empirical results show that the proposed simulated annealing algorithm with random and swapping neighbourhood structures outperforms the other algorithms in terms of makespan. However, the proposed approach needs to be compared with other metaheuristics algorithm. In addition, more experiments using dynamic grid computing environment are required to test the algorithm robustness. Nonetheless, the idea of using various neighbourhood structures is very important for local search algorithms such as simulated annealing.

An implementation of simulated annealing algorithm for job scheduling in computational grid has been proposed by Guo and Wang (2010). The proposed approach tried to overcome two problems in implementing SA algorithm in grid

computing, namely the algorithm overhead and the best tuning parameters. The authors developed a simulator called Ana-GridSim which is an extension of GridSim simulator developed by Buyya and Murshed (2002). The experiment results show that the proposed approach achieved good performance in terms of average error. However, the experiment design was very poor in terms of benchmark problems (only 7 resources were used). In addition, optimization function was not defined, such as makespan values have not been optimized. Moreover, the proposed algorithm was not compared with other metaheuristics algorithms. Therefore, a more comprehensive investigation on implementing simulated annealing algorithm for job scheduling on grid computing systems is required.

II. Tabu Search

Tabu search is a metaheuristics algorithm based on a guided local search developed by Glover (1986). TS algorithm has been successfully implemented to solve many optimization problems, such as job shop scheduling, travelling salesman problem, and vehicle routing problem (Gendreau & Potvin, 2014). According to Burke and Kendall (2014), over the last 25 years, hundreds of papers presenting applications of tabu search proposed to solve various combinatorial problems. TS is classified as a local search that has the ability to skip from the local minimum by applying many mechanisms such as memory and diversification (Rothlauf, 2011). Specifically, TS applies the concept of adaptive memory and responsive exploration that make the algorithm more flexible. TS algorithm operates using four types of memory, namely: recency (short-term memory), frequency (long-term memory), quality, and influence (Glover & Laguna, 1997). However, many successful applications use only one or two of these memories. Figure 2.5 shows the tabu search algorithm process.

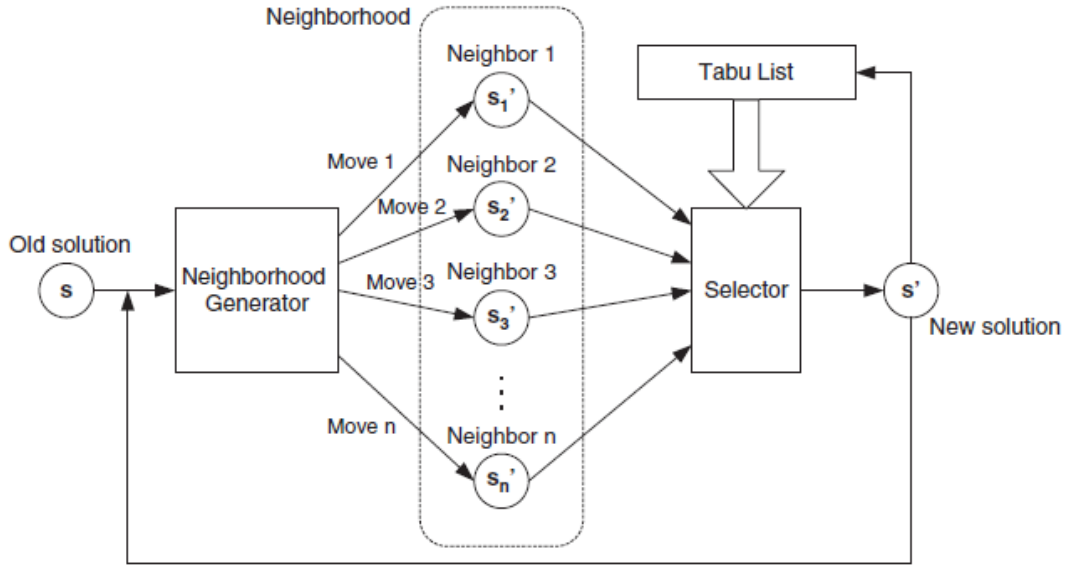

Figure 2.5. Process of Tabu Search algorithm (Zapfel et al., 2010)

The memory in tabu search could be explicit or attributive. Explicit memory records the complete solution, while attributive memory records the information about the solution attribute that changes when moving from one solution to another. For example, in job scheduling scenario, moving a task t_i from machine m_p to machine m_q will create a new solution vector. Hence, TS memory could be implemented based on recording the complete old solution or recording only the attribute that changed the solution that is the history of assigning task t_i to machine m_p. Recording the attributes will prevent the algorithm (tabu) to reverse to the old assignment for k number of iterations with the same task and machine. However, this tabu attribute could be override if the move will produce a solution better than the best-so-far solution (could be any other criteria); this mechanism is called *aspiration level*. The duration parameter for a move to be considered as a tabu is called *tabu tenures* (Glover & Laguna, 1997). Effective tabu tenures depend on the size of a problem instance.

Tabu search starts with initial solution which could be created randomly or by using any ad-hoc algorithm such as Max-min, Min-min algorithm for scheduling (Xhafa, Carretero, et al., 2008). From the initial solution, TS will start searching the neighbourhood in order to find the local optima. If the move to the neighbour solution is not tabu, then the neighbour solution is saved as the current solution. If the neighbour solution is better than the best-so-far solution, then it saved as the best-so-far solution. In the case that the move is tabu, the inspire level will be checked in order to override the tabu if the neighbour solution is better than best-so-far solution. After moving to the neighbourhood solution, TS will update its memory and start a new iteration if the termination condition is not met. The mechanism of searching the neighbourhood repeatedly seems to guide the search towards an interesting area in the search space quickly (Costa, 1994). However, there are many issues which need to be addressed when implementing tabu search, such as what information to be saved in the memory, the size of the tabu lists, how to move to the neighbourhoods, and how to implement diversification (Thesen, 1998). Tabu search algorithm is applied for scheduling problems in grid computing successfully. Figure 2.6 shows tabu search algorithm pseudocode.

Procedure Tabu Search Algorithm
Step 1- Create initial solution s;
Step 2- Create global solution $s^* \leftarrow s$;
Step 3- Create tabu list TL;
Step 4- Initialize the aspiration function A;
Step 5- **While (termination condition not satisfied) Do**
Step 6- Search the neighbourhood N of current solution s: $\{\hat{s} \in N(s)\}$;
Step 7- If (move from s to $\hat{s}$ is not in TL) Then
Step 8- $s \leftarrow \hat{s}$;
Step 9- End If;
Step 10- Else If $(f(\hat{s}) < A(f(s))$ Then
Step 11- $s \leftarrow \hat{s}$;
Step 12- End If;
Step 13- Update TL memories;
Step 14- If $(f(s) < f(s^*))$ Then
Step 15- $s^* = s$;
Step 16- End If;
Step 17- **End While;**
Step 18- Return Global solution s^*;
End Procedure;

Figure 2.6. Tabu Search algorithm pseudocode (Zapfel et al., 2010)

In Figure 2.6, the first step in TS algorithm is creating the initial solution using random approach or ad hog algorithm. The second step is initializing the global solution from the initial solution. Third step is creating the tabu list to store the movement history. The forth step in initializing the aspiration function to be used to override the tabu movement.

Fifth step is the starting of the algorithm iteration which is terminated using stop condition, such as the algorithm reached the maximum number of iterations, reach the maximum time allowed, or no enhancement achieved for specific number of iterations.

Sixth step is searching the neighbourhood in order to generate different solution which is based on current solution (Glover & Laguna, 1997). Step seven will check if the movement to the new neighbourhood solution is not tabu (not visited before) then the solution will be accepted even if it is worse than the current solution. This technique makes TS algorithm skips from local optima trap. Step eight will save the new accepted solution as the current solution and the condition ends at step nine. If the solution is tabu, step ten will check the aspiration function to determine whether to override the tabu if the solution quality better than the current solution or discard the new solution. Step eleven will save the new solution as the current solution if the aspiration condition satisfied and the step will end in step twelve.

Step thirteen will update the tabu list memory, such as the short and long memories. Step fourteen will check if the current new solution is better than the global solution, then the current solution will be saved as the globe best solution in step fifteen and ends in step sixteen. In job scheduling problem, the solution quality is measured using makespan metric. Step seventeen will end the while and the best global solution return in step eighteen.

Tabu search design and evaluation for job scheduling in grid computing has been proposed by Xhafa, Carretero, et al. (2008). The aim of the study is to minimize the makespan and flowtime values as a bi-objective optimization problem. The bi-objective function is implemented with a hierarchic approach in which makespan is considered as a primary objective and flowtime as a secondary objective. The proposed algorithm starts with the initial solution generated using Min-Min algorithm. To search the neighbourhood of the initial solution, two types of movements were implemented, namely transfer and swap which are adopted from Thesen (1998). The

proposed study used the expected time to compute model as developed by Braun et al. (2001). The implemented algorithm was compared versus tabu search and ant colony optimization algorithm hybridized with tabu search as proposed in Ritchie and Levine (2004). The experiment results show that the proposed design and implementation outperforms the approach proposed by Ritchie and Levine (2004). Another experiment was conducted using the benchmark problem as defined by Carretero and Xhafa (2006) which represent a large instance problem in dynamic environment. The proposed algorithm was compared against steady-state genetic algorithm developed by Carretero and Xhafa (2006). Again, the results show that the proposed algorithm achieved better makespan values than steady-state genetic algorithm. The authors concluded that the proposed design and implementation for tabu search algorithm are more efficient than other implementation. In addition, the authors noticed that makespan and flowtime are contradictory objectives; this observation is very important to understand the complexity of job scheduling in grid computing.

Tabu search algorithm has been implemented for job scheduling in grid computing by Xhafa, Carretero, Dorronsoro, and Alba (2009). The authors defined a bi-objective optimization problem consisting of makespan and flowtime. Their proposed algorithm adapted two types of neighbourhood movement namely transfer and swap. In addition, the algorithm implanted intensification and diversification strategies to achieve better results. Their study dealt with static and dynamic environments for algorithm evaluation. For static environment, the benchmarks were generated based on expected time to compute model as presented by Ali, Siegel, Maheswaran, Hensgen, and Ali (2000a). While for dynamic environment, the benchmarks were generated using extended HyperSim simulator as presented in Phatanapherom, Uthayopas, and Kachitvichyanukul (2003). The proposed algorithm was compared

with three metaheuristics algorithms, namely TS, ACO+TS, and cMA (Ritchie & Levine, 2004; Xhafa, Alba, et al., 2008) for static experiment. For dynamic experiment, the proposed algorithm was compared with GA as presented by Carretero and Xhafa (2006). The experiment results show that the proposed algorithm outperforms the other algorithms in static and dynamic environment. The study provides all the implementation details and the pseudo-code as well which makes the study repeatable and easy to re-implement. However, there are many neighbourhood movements other than transfer and swap, which has the ability to find better local optima, such as insert and load balance moves.

2.1.4 Swarm Intelligence Algorithms

According to Gazi and Passino (2011), the terminology of "swarms" has come to mean as "a set of agents possessing independent individual dynamics but exhibiting intimately coupled behaviours and collectively performing some task". Swarm Intelligence (SI) algorithms are these algorithms which are nature-inspired such as ant colony optimization, artificial bee colony, particle swarm optimization, cuckoo search, and firefly algorithms (Yang, 2014). SI algorithms try to mimic the biological behaviour of some creatures such as colonies of ants or bees, flocks of birds, and schools of fish (Gazi & Passino, 2011). Swarm intelligence algorithms are inspired the field of computing study, specifically the optimization field (Pintea, 2014). In terms of computational model, swarm intelligence models are considered as computing algorithms that are useful for solving distributed optimization problems (Lim & Jain, 2009). The principles of swarm intelligence algorithm are proximity, quality, diverse response, stability, and adaptability (Lim & Jain, 2009). Swarm intelligence methods have shown very successful performance in the area of

scheduling which is of great importance for the industry and science (Blum & Li, 2008). The following subsections discuss the studies of swarm intelligence algorithms for job scheduling in grid computing.

I. Ant Colony Optimization

In 1992, Marco Dorigo presented the first ACO algorithm in his PhD thesis to search for an optimal solution in graph (Dorigo & Stutzle, 2004).The variants of ACO are:

(i) Ant System (AS), ant system is the first algorithm introduced in ant colony optimization algorithms and the prototype of a number of ant algorithms extension. It was initially proposed by Colorni, Dorigo, and Maniezzo (1991), and Dorigo et al. (1991) aimed to search for an optimal path in a graph based on the behaviour of ants seeking a path between their colony and food source. AS is also the first ACO algorithm which has been applied to the travelling salesman problem (Dorigo et al., 1996). Three different versions of ant system were proposed, which are ant-density, ant-quantity, and ant-cycle. In ant-density and ant-quantity, the ants update the pheromone directly after a move from a city to another city. But in ant–cycle, the pheromone update was only done after all the ants had constructed the tours. The two main phases of the ant system algorithm constitute the ants' solution construction and the pheromone update. The performance of ant system when compared to other algorithms tends to decrease dramatically as the size of the test-instances increases. For AS tour construction, an ant applies probabilistic action choice rule, called *random proportional* rule, to decide which node to visit next. The probability of the ant to move from node to node depends on pheromone and heuristic values. AS updates the pheromone trails after all ants have constructed their tours. The first step in pheromone updating is lowering the pheromone values (*evaporation*) on all arcs by

a constant factor. This step will enable the algorithm to forget the bad decision previously taken, at the same time if the arc is not chosen by the ants, its associated pheromone value decreases exponentially in the number of iterations. After evaporation, all ants deposit pheromone on the arcs they have visited in their tour.

(ii) The first improvement on ant system, called the Elitist strategy for Ant System (EAS), was introduced by Dorigo et al. (1991, 1996). This algorithm provides strong additional reinforcement to the arcs belonging to the best tours found since the start of the algorithms. In EAS, the global best solution deposits pheromone on all iterations along with all the other ants and the pheromone evaporation is implemented as in the ant system. The use of the elitist strategy allows the ant system to both find the better tours and find them in a lower number of iterations. The additional reinforcement of best tour is achieved by adding an extra quantity of pheromone to its arcs based on the tour length and a new parameter defined as weight is given to the best-so-far tour.

(iii) Rank-Based Ant System (AS_{rank}), another improvement over the ant system is the rank-based version of ant system introduced by Bullnheimer, Hart, and Straub (1999). In AS_{rank}, each ant deposits an amount of pheromone that decreases with its rank. In addition, as in EAS, the best-so-far ant always deposits the largest amount of pheromone with during iteration. In AS_{rank}, the first step for updating the pheromone trails is sorting the ants by increasing the tour length. The quantity of pheromone an ant will deposit is weighted according to the rank of the ant. During iterations, only the best ranked ants and the ant that produced the best-so-far tour are allowed to deposit pheromone. Among the AS-based algorithms, both, AS_{rank} and EAS performed significantly better than AS, with AS_{rank} giving a slightly better result than EAS.

(iv) Another variant of ACO algorithm is Max-Min Ant System (MMAS). This algorithm has direct improvement over AS (Stutzle & Hoos, 1997, 2000). MMAS differs from the basic approaches of AS in the following aspects. Firstly, it uses a greedier search mechanism that allows a good exploitation of the accumulated experiences. Secondly, MMAS uses a range of pheromone trail values to the interval that help to avoid the premature stagnation (all ants converge early to one suboptimal solution) of the search process. Thirdly, the initial value of pheromone trails is set to the upper pheromone trail limit with a small pheromone evaporation rate to increase the exploration of tours at the start of the search. Finally, in MMAS, pheromone trails are reinitialized each time when the system does not produce an improved tour for a certain number of consecutive iterations. To update pheromone trails, in MMAS, after all ants have constructed a tour, pheromones are updated by applying evaporation as in AS. After that, the deposit of a new pheromone is applied based on the best-so-far tour. Only one of the two ants is allowed to add pheromone, either the best-so-far ant or the iteration-best ant. In MMAS, lower and upper limits [$\tau_{\min}$ and $\tau_{\max}$] of pheromone on any arc are used to avoid the search stagnation.

(v) Ant Colony System (ACS), this improvement has been introduced by Dorigo and Gambardella (1997a, 1997b) to improve the performance of AS. ACS differs in three main aspects from ant system. First, ACS uses a more aggressive action choice rule than AS. Second, the pheromone is added only to moves belonging to the global-best solution. Third, each time an ant moves on a path, it removes some pheromone from that path. The three main phases of the ACS algorithm constitute the ants' solution construction, global pheromone trail update, and local pheromone trail update. ACS algorithm starts solution construction when the ant moves from node to node. The ant will choose the node using one of the two rules. The first rule is called *pseudorandom*

proportional rule which is based on exploitation mechanism. The second rule uses exploration mechanism which is based on the probability distribution used in AS. The tuning between exploitation and exploration is controlled by a parameter fixed by the user. ACS algorithm applies the global pheromone trail update. In this update, only one ant (the best-so-far ant) is allowed to add pheromone after all ants have finished constructing their tours. In addition, ACS algorithm applies the local pheromone trail update. In this update, all the ants apply a local pheromone update rule immediately after moving on arcs during the tour construction using the evaporation concept.

In ACS algorithm, when ant *k* wants to move from node *i* to node *j*, it will choose the node using a rule called *pseudorandom proportional* rule, calculated as:

$$P_{ij}^{Ant_k} = \begin{cases} argmax_{l \in N_i^k} \{\tau il[\eta il]^\beta\}, & \text{if } q \leq q0; \\ J, & \text{otherwise} \end{cases} \quad (2.2)$$

where q is a random variable uniformly distributed in [0, 1], q0 ($0 \leq q0 \leq 1$) is a parameter, and J is a random variable selected according to the probability distribution calculated as:

$$J = \frac{[\tau ij]^\alpha [\eta ij]^\beta}{\sum_{l \in N_i^k} [\tau il]^\alpha [\eta il]^\beta}, if\ j \in N_i^k \quad (2.3)$$

with α = 1. The tuning between exploitation and exploration is controlled by the parameter $q0$.

In ACS only one ant (the best-so-far ant) is allowed to add pheromone after each cycle. The Update is implemented using the following equation:

$$\tau ij \leftarrow (1 - P)\tau ij + p\Delta\tau_{ij}^{bs}, \quad \forall (i,j) \in T^{bs}, \tag{2.4}$$

Where P is the pheromone evaporation rate, and $\Delta\tau_{ij}^{bs} = 1/ C^{bs}$.

For local pheromone update, a rule immediately applied after moving on arc (*i, j*) during the solution construction using the following equation:

$$\tau ij \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0, \tag{2.5}$$

Where ξ, $0 < \xi < 1$, and τ_0 are two parameters. The value of τ_0 is equal to the initial value for the pheromone trail.

Each variant of ACO algorithms implemented construction and pheromone update methods. However, there are several differences between them. Table 2.1 shows the differences between each variant in ACO algorithms.

Table 2.1

Difference between each variant algorithm in ACO

Algorithm name	**Differences and work mechanisms**
AS	All ants deposit pheromone on the arcs they have crossed in their tour.
EAS	Provides strong additional reinforcement to the arcs belonging to the best tour by adding a quantity e/C^{bs}.
AS_{rank}	Ant deposits an amount of pheromone that decrease with its rank. In each cycle, only the best-so-far ant always deposits the largest amount of pheromone.
MMAS	It strongly exploits the best tour found. Limits the possible range of pheromone trail values to the interval $[\tau_{min}, \tau_{max}]$. The pheromone trails are initialized to the upper pheromone trail limit. Pheromone trails are reinitialized each time the system approaches stagnation.
ACS	It exploits the search experience accumulated by the ant more strongly than AS.

	Pheromone evaporation and pheromone deposit take place only on the arcs belonging to the best-so-far tour. Each time an ant uses the arc (*i, j*) to move from node *i* to node *j*, it removes some pheromone from the arc to increase the exploration.

ACO algorithm has been applied on several domains such as optimization, classification, bioinformatics, and network.

Some domains contain more than one problem that needs to be solved using ant colony optimization. Table 2.2 presents several types of researches conducted on different domains.

Table 2.2

Various research on different Domains and problems

Domain	Problem Name	ACO type	References
Routing	Travelling Salesman Problem	Novel ACO	(Zhu, Zhao, & He, 2010b)
	Vehicle Routing Problem	ACS	(Calvete, Gale, & Oliveros, 2012)
	Vehicle Scheduling Problem	Hybrid AS	(Zhang, Ning, & Zhang, 2012)
Grid Computing	Task Scheduling	AS	(Wei et al., 2012)
	Resource Discovery	AS	(Devi & Pethalakshmi, 2012)
	Grid Resource Scheduling	Improved AS	(Liu, Ma, Guo, & Wang, 2012)
Image Processing	Image Edge Detection	AS	(Tian, Yu, Chen, & Ma, 2011)
	Image Classification	AS	(Seo, 2012)
	Optic Disc Detection	AS	(Pereira, Goncalves, & Ferreira, 2013)
Operational Research	Sequential Ordering Problem	Enhanced ACS	(Gambardella, Montemanni, & Weyland, 2012)
	Surgery Scheduling Problem	AS	(Yin & Xiang, 2012)
	Process Planning Optimization	AS	(Liu, Yi, & Ni, 2013)
Manufacturing	Assembly Sequence Planning	MMAS	(Yu & Wang, 2013)
	Assembly Sequence	AS	(Youhui, Xinhua, & Qi,

	Planning		2012)
	Assembly Line Balancing Problem	AS	(Zheng, Li, Li, & Tang, 2013)
Database	Distributed Join Query Optimization	MMAS	(Golshanara, Rankoohi, & Shah-Hosseini, 2013)
	Optimization For RDF Chain Queries	AS	(Hogenboom, Frasincar, & Kaymak, 2013)
	Optimization Of Distributed Database Queries	Hybrid AS	(Dokeroglu & Cosar, 2012)
Electrical Engineering	Control Of Ocean Wave Energy	AS	(See, Tai, & Molinas, 2012)
	Bus Priority In Power System	AS	(Hamid, Musirin, Rahim, & Kamari, 2012)
	Power Signal Pattern Classification	Hybrid AS	(Biswal, Dash, & Mishra, 2011)
Data Mining	Classification Rule Discovery	AS	(Hodnefjell & Junior, 2012)
	Data Classification	ACS	(Michelakos, Mallios, Papageorgiou, & Vassilakopoulos, 2011)
	Classification And Rule Generation	AS	(Tiwari & Verma, 2012)
Bioinformatics	Epistasis Detection	AS	(Shang, Zhang, Lei, Zhang, & Chen, 2012)
	Finding Optimal Spaced Seeds	AS	Duc, Dinh, Dang, Laukens, & Hoang, 2012)
	Classifying Imbalanced DNA	AS	(Yu, Ni, & Zhao, 2013)
Robotics	Robot Path Planning	AS	(Chen, Kong, Fang, & Wu, 2011)
	Robot Path Planning	AS	(Bai, Chen, Jin, Chen, & Mao, 2012)
	Multi-Tasks Distribution In Heterogeneous Robot	AS	(Lope, Maravall, & Quinonez, 2012)
Networks	Energy-Saving Routing For Wireless Sensor Networks	AS	(Chen, Yu, Hong, & Dong, 2012)
	Routing For Hierarchical Wireless Sensor Networks	AS	(Wang, Jing, & Wang, 2012)
	Routing And Spectrum Allocation	AS	(Wang, Zhang, Zhao, Wang, & Gu, 2013)
Assignment	Timetabling Problem	AS	(Nothegger, Mayer, Chwatal, & Raidl, 2012)
	Graph Colouring Problem	AS	(Douiri & Elbernoussi, 2012)
	Aircraft Assigning	ACS	(Zhang, Lin, Qiu, & Fu,

Problem	2011)

Various AS algorithms have been derived and extended to exploit the search history without losing the chance of exploring new areas of the search space. Among them, ACS algorithm appears to be promising to extend the framework of ACO. It provides a good opportunity to explore a wide area of the search space in reasonable time. All variants of ACO algorithm have some similarity in their foundation, such as utilizing the heuristic information, pheromone value, and the solution is based on constructing. All ACO algorithms apply pheromone evaporation.

Cordon, Viana, and Herrer (2002) and Cordon, Viana, Herrera, and Moreno (2000) proposed the Best-Worst Ant System (BWAS) as another extension of the basic idea of AS by including some concepts from evolutionary computation algorithms. BWAS uses the same transition rule as in AS algorithm to construct ants' solutions. Then, BWAS enhances the ants' solution by using a local optimizer to bring each solution to its local optimum. Like AS, AS_{rank} and MMAS, all pheromone updates are performed by daemon actions.

Lorpunmanee, Sap, Abdullah, and Chompoo-inwai (2007) presented a study called "An Ant Colony Optimization for Dynamic Job Scheduling in grid Environment". In their study, they developed a general framework of grid scheduling using dynamic information and an ant colony optimization algorithm to improve the decision of scheduling. The experiment was conducted using GridSim simulator toolkit version 4.0 with an extension. They presented a comparison between ant colony optimization and other various algorithms for job scheduling and dispatching rules for grid environment, such as First Come First Served (FCFS), Minimum Time Earliest Due

Date (MTEDD), and Minimum Time Earliest Release Date (MTERD). Their experiment results stated that ACO is able to perform the optimal scheduling job. Besides that, the ACO accounts for less than 17% of the total tardiness time in the average when it is compared with the other scheduler algorithms. However, in their approach, ACO algorithm performs much slower than the other scheduler algorithms.

A research was conducted on Balanced job scheduling using Ant Colony Optimization (BACO) for grid environment by Chang, Chang, and Lin (2007). The main issue they tried to solve is how to schedule jobs efficiently in a grid environment. In their approach, they used four main components: portal, information server, jobs scheduler, and grid resource. BACO algorithm applied inside jobs scheduler in order to select the most appropriate resource to execute the job. The experiment was implemented using Taiwan UniGrid platform which consists of 26 campuses. BACO performance was compared with other two algorithms: Improved Ant Algorithm (IACO) and Fastest Processor to Largest Task First Algorithm (FPLTF). The results show that BACO has the ability to balance the job scheduling load in the entire system. However, according to Bai et al. (2010) and Liu, Song, and Dai (2010), using the single ant colony system leads to local optima because of the stagnation that occurs due to the positive pheromone feedback mechanism.

A study proposed by Chang, Chang, and Lin (2009) implemented ant algorithm for balanced job scheduling in computational grid. The study aimed to balance the entire system load, at the same time tried to minimize the makespan of the set of jobs. In addition, the study considered the bandwidth speed between the scheduler and resources as well. The proposed algorithm is based on ant system algorithm. The study was implemented in the Taiwan UniGrid platform which consists of more than

20 campuses in Taiwan. The experiments also simulated two problems, matrix multiplication and linear programming. The evaluation was done by comparing it with the improved ACO algorithm as proposed by Yan, Shen, Li, and Wu (2005), the fastest processor to the largest task first algorithm (Menasce, Saha, Porto, Almeida, & Tripathi, 1995), suffrage algorithm (Silva, Cirne, & Brasileiro, 2003), and random method. In terms of makespan and balance, the experiment results show that the proposed algorithm outperforms the other algorithms. However, the experiments are very limited and unrepeatable due to the using of real grid computing environment. Moreover, the comparison did not include other metaheuristics algorithms such as genetic algorithm and artificial bee colony. Nevertheless, the study provides a practical implementation guide which is useful to illustrate how job scheduling in computational grid is working.

Kousalya and Balasubramanie (2009) presented a study on improving ant colony optimization algorithm with local search for job scheduling in computational grid systems. The proposed approach aims to minimize the makespan value as the main objective. The study adopted local search technique from Ritchie and Levine (2003) which is based on several neighbourhood searches such as Swap, Move, and Move Top methods. The experiments were conducted using static benchmark problems generated using Execution Time (ET) model presented by Fidanova (2006). The proposed algorithm was evaluated versus Min-Min, ACO, Swap, Move, and Move Top algorithms. The experiment results show that using ant colony optimization algorithm with Move Top local search method outperforms other algorithms in terms of makespan. However, the proposed algorithm was not compared with other metaheuristics and hybrid algorithms in order to prove the algorithms performance. In

addition, more experiments using dynamic grid computing environment are required which could provide more understanding to the behaviour of the proposed algorithm.

Liusuqin, Shuojun, Menglingfen, and Lixingsheng (2009) proposed a study to improve ant colony optimization for Job Scheduling Problem (JSP). In their approach, they address the problem of "misusing the great resources for minor purpose" which makes some resources always idle and some resources are busy processing jobs. They solved the problem by introducing improved ACO algorithm called "making concessions in order to gain advantages" based on ACO. The experiment was conducted using grid pheromone model simulation developed by the authors. The proposed algorithm was compared with ant colony optimization using static benchmark problems. The results show that the improved ACO algorithm could perform better than the conventional ACO. The new algorithm could make better use of the resources and solve the "misusing the great resources for minor purpose" problem. However, the experiment size is very limited and did not contain heterogeneous resources and tasks. In addition, the proposed algorithm was not compared with other types of metaheuristics algorithms such as genetic algorithm and tabu search algorithm.

A multiple ant colony model called "Cooperative multi-ant Colony Pseudo-parallel Optimization Algorithm" was proposed by Liu et al. (2010). In their approach, three subcolonies were used for optimization. Each subcolony respectively uses ant system algorithm, ant colony system algorithm, and max-min ant system algorithm independently. Each subcolony has its own pheromone matrix. By using three different colonies, the pheromone matrix will have different distributions and characteristics of change. After going through a certain number of iterations and

fulfilling the condition of pheromone interaction, the matrices of pheromone of the three subcolonies will interact. The interaction will be according to their weight value to gain new pheromone matrix. After the interaction process, the algorithm re-initializes the three pheromone matrices. They conducted experiments to solve the travelling salesman problem. The experiment results show that their algorithm performance is better than the classic algorithms (AS, ACS, and MMAS). The researchers of the model claim the ability to prevent the system from stagnation because of different distribution of different ant algorithm used. However, their algorithm performs slightly better than classical MMAS, while MMAS has less complexity in implementation and processing time.

Another study regarding ACO algorithm for job scheduling on computation grid has been proposed by Kant, Sharma, Agarwal, and Chandra (2010). The authors proposed two types of ant, namely red ants and black ants. The red ants' responsibility is system resource estimation, while the black ants' responsibility is decision of resource allocation. The study objective is to minimize the maximal total tardiness time of all jobs within the machines in grid environment. The proposed approach was simulated in real grid environment using 49 different resources. The comparative study was done using Min-Min and FCFS algorithms (Fidanova & Durchova, 2006; K. Liu, Chen, Jin, & Yang, 2009). The experiment results show that the proposed algorithm outperforms other algorithms. However, the experiments are very limited and the comparison did not include other metaheuristics algorithms. Therefore, more investigations are required to test the algorithm robustness.

Load balancing is one of the important criteria in grid computing. A research on task scheduling with load balancing using Multiple Ant Colonies Optimization (MACO) in

grid computing was conducted by Bai et al. (2010). In their work, they used multiple ant colony optimizations to avoid the local optima from single colony behave. In their framework, they considered both positive and negative feedbacks in searching for solutions by sharing the search information and exploring a wider area of search space with the cooperation between the ant colonies. They defined the degree of imbalance by calculating the heuristic value using the load computing of each node. The experiments were conducted to evaluate the proposed algorithm with benchmark problems generated using GridSim simulator developed by Buyya and Murshed (2002). They compared MACO with ant colony systems (Dorigo & Gambardella, 1997b) and first-come-first-served (Harchol-Balter, Crovella, & Murta, 1999). The results showed that their algorithm outperforms other algorithms in terms of makespan and load balancing. However, the solution for intractability between performance and load balancing is not illustrated.

Enhanced ant colony algorithm for job scheduling in computational grid has been proposed by Maruthanayagam and UmaRani (2010). The proposed approach is based on Fast Ant System (FANT) algorithm which is a version of ACO algorithm. The study focused on makespan optimization using the independent task model as defined in Kousalya ad Balasubramanie (2009). The authors compared between two formulas proposed to calculate the probability of selecting a resource for processing a task. The experiment results show that using local search algorithm will improve the algorithm performance significantly. However, the study lacks a proper experiment design such as using known benchmark problems and comparing the proposed algorithm with other metaheuristics algorithms. Nonetheless, using fast ant system could be suitable for job scheduling in grid computing due to the time restriction imposed by computational grid systems.

Mou (2011) proposed a new approach using double Pheromones techniques for ant colony system. The study model was designed to solve Generalized Travelling Salesman Problem (GTSP) which is an extension of the classical traveling salesman problem. In GTSP, the nodes were partitioned into groups called clusters. The solution to GTSP is to find the shortest closed tour visiting exactly one node from each cluster. For such a problem, there are two pheromones, namely pheromone between the groups and pheromone on the edges. The researcher tried to differentiate between those pheromones by applying the double pheromones concept. In addition, a mutation idea inspired from genetic algorithm was introduced in this study. According to the experiment results conducted by the author, applying double pheromones produced better performance. However, the instances used in the experiment were small. According to Li, Liao, and Cai (2011), they stated about ant colony system that "it is difficult to realize the overall optimum and it takes a long time when being applied to large-scale TSP". In addition, the implementation and influence of the mutation idea was not illustrated in the study.

An improved ACO algorithm for job scheduling in computational grid systems has been proposed by MadadyarAdeh and Bagherzadeh (2011). The main objective of the study is to minimize the makespan value as an optimization objective in a batch mode. The authors improved the ACO algorithm for job scheduling provided in Kousalya and Balasubramanie (2008). The improvement is based on giving higher probability to tasks that have higher standard deviation. For evaluation purpose, the study adopted static benchmark problems based on expected time to compute model using Range-Based method as proposed by Ali et al. (2000b). The proposed algorithm was compared with ACO and P.ACO presented in Bagherzadeh and MadadyarAdeh (2009). The experiment results show that the proposed algorithm achieved the best

makespan values among other algorithms. However, the experiments were conducted using a very small scenario (32 tasks and 4 machines) which is not sufficient to test a metaheuristics algorithm. In addition, only static environment were considered in the experiment without any dynamic features to reflect the real job scheduling problem in grid computing. Moreover, the proposed algorithm was only compared with ACO approaches. Therefore, more comprehensive experiments and comparisons are required in order to discover the efficiency of the proposed algorithm.

Kokilavani and Amalarethinam (2013) published a study on implementing ant colony optimization based load sharing for job scheduling in computational grid systems. The study aims to enhance the quality of service and share the load among the resources in order to optimize the resource usage in the computational grid environment. The proposed algorithm was implemented in MATLAB application simulating grid computing with 2 resources and 5 tasks. The proposed ant colony optimization based load sharing was compared with Min-Min and Max-Min algorithms. The experiment results show that the proposed approach outperforms other algorithms in terms of total wait time criterion. However, the experiments are very limited in terms of the benchmark problem size. In addition, the study is based on static environment and used unknown scenario. Moreover, the proposed algorithm was only compared with simple heuristic approaches. Therefore, the proposed algorithm should be evaluated on dynamic environment and compared with other metaheuristics algorithms.

II. Artificial Bee Colony

Bee algorithms are inspired by biological honeybee behavioural specifically the foraging and exploration (Yang, 2014). There are several types of bee algorithms, such as honeybee algorithm, virtual bee algorithm, artificial bee colony, and

honeybee-mating algorithm. Artificial Bee Colony (ABC) is an optimization algorithm developed by Karaboga and Basturk (2008) and Karaboga (2005). The bees in ABC algorithm are divided into three groups, namely: employed bees, onlooker bees, and scout bees. The idea behind ABC is that for each food source, there is only one employed bee. In other words, the total number of employed bees is equal to the total number of food sources. When the food source is discarded, the employed bee of that food source is forced to be a scout bee. The scout bee will search for new food source randomly. The onlooker bees wait in the hive to obtain the information from the employed bees. Based on that information, the onlooker bees will choose the best food source probabilistically and start foraging (Yang, 2014). ABC algorithm is applied to solve job scheduling problem in computational grid.

A recent study published by Kim, Byeon, Liu, Abraham, and McLoone (2013) applied Artificial Bee Colony (ABC) for job scheduling in computational grid. The authors proposed Binary ABC (BABC), Efficient Binary Artificial Bee Colony (EBABC1), and flexible ranking strategy (EBABC2) algorithms. The study aimed to minimize the makespan criterion for job scheduling in grid computing. The experiments were conducted using a series of benchmark problems defined by Liu et al. (2010). The proposed algorithms were compared with genetic algorithm, simulated algorithm, and particle swarm optimization algorithm. In terms of makespan criterion, EBABC1 and EBABC2 algorithms achieved the best results among all other algorithms with superior performance for EBABC2. However, the experiments were conducted using static environment which is not enough to conclude the algorithm robustness in dynamic environment. Therefore, conducting more experiments is required. In addition, hybridizing artificial bee colony with local search seems a promising research area as well.

III. Bacterial Foraging Optimization Algorithm

Bacterial algorithms mimic the behaviour of bacteria in the nature such as foraging, reproduction, and movement (Xing & Gao, 2014). There are several types of bacterial algorithms, such as bacterial foraging algorithm, bacterial colony chemotaxis, superbug algorithm, bacterial colony optimization, and viral system (Xing & Gao, 2014). Bacterial algorithms have been implemented successfully in various scheduling problems such as job shop scheduling problems (Ge & Tan, 2012; Wu, Zhang, Jiang, Yang, & Liang, 2007), flow shop scheduling problems (Botzheim, Toda, & Kubota, 2012), and assembly line balancing (Atasagun & Kara, 2014). Bacterial algorithms have been utilized to solve job scheduling problems in grid computing systems.

Nayak, Padhy, and Panigrahi (2012) proposed an algorithm which combined the merits of genetic algorithm and bacterial foraging optimization algorithm called Genetic Bacterial Foraging (GBF). The proposed algorithm implemented a dynamic mutation as presented in Michalewicz (1999) and crossover operator developed by Michalewicz (1992). The aim of the study is to reduce the execution time as a cost function. The experiment was conducted using dynamic environment generated with a simulator developed by the authors. The proposed algorithm was compared with Bacterial Foraging Optimization (BFO) algorithm. The experiment results show that the proposed GBF algorithm outperforms BFO algorithm. However, the experiment scenario was very small, using only four resources and five tasks. Therefore, more studies are required to understand the behaviour of bacterial foraging optimization algorithm.

A study has been conducted by Rajni and Chana (2013) on Bacterial Foraging optimization (BFO) algorithm for resource scheduling on computational grid systems. The study aimed to optimize makespan and cost values by considering Resource Provisioning (RP) unit adopted from Aron and Chana (2012). The proposed approach was implemented using GridSim simulator developed by Buyya and Murshed (2002). The experiments were conducted by generating a workload using a model defined in Lublin and Feitelson (2003) and expected time to compute model presented in Ali et al. (2000a). The authors compared the proposed algorithm with genetic algorithm, simulated annealing, and GA-TS algorithms. The experiment results show that the proposed BFO algorithm outperforms other algorithms in terms of makespan and cost values for both low and high machine heterogeneity benchmark problems. In addition, the results show that the Coefficient of Variation (CV) of the proposed algorithm is in the range 0%-2% which confirms the stability of the proposed algorithm. However, the experiments are very limited and did not include some dynamic grid attributes such as resource failure which is considered very important in dynamic grid computing system (Feitelson, 2013).

IV. Particle Swarm Optimization Algorithm

Particle Swarm Optimization (PSO) algorithm was initially developed by Eberhart and Kennedy (1995) and Kennedy and Eberhart (1995). PSO is considered as a population-based optimization algorithm based on biological swarm intelligence (Noghanian, Sabouni, Desell, & Ashtari, 2014). PSO has been implemented to solve many real time problems such as face recognition (Kothari, Anuradha, Shah, & Mittal, 2012), assembly scheduling problem (Tian, Liu, Yuan, & Wang, 2012), Resource-Constrained Project Scheduling Problem (Jia & Seo, 2013), and job shop

scheduling problem (Li & Pan, 2012). PSO is applied successfully to solve job scheduling problems in computational grid system.

In job scheduling problems for grid computing environment, a fuzzy PSO approach was published by Abraham, Liu, Zhang, and Chang (2006). The proposed algorithm extends the position and velocity of the particles from real vectors to fuzzy matrices. The study aimed to optimize the scheduler performance in terms of makespan and flowtime as a bi-objective. The performance of the proposed algorithm was evaluated with genetic algorithm and simulated annealing approaches. The experiments were conducted using three static instances generated by the authors. The evaluation results show that the proposed fuzzy PSO algorithm was able to achieve better makespan values than other algorithms. However, the results did not include flowtime values which are supposed to be the second objective of the study. Thus, no conclusion could be provided regarding the performance of the proposed algorithm. Moreover, the experiments were conducted using only static scenario which is not enough to explore the proposed algorithm behaviour.

Izakian, Abraham, and Snasel (2009a) proposed a particle swarm optimization algorithm for meta-tasks scheduling in distributed heterogeneous computing systems. The proposed approach aims to minimize makespan as an objective function. The implemented PSO algorithm was compared with genetic algorithm as presented in Braun et al. (2001), and continuous PSO developed by Salman, Ahmad, and Al-Madani (2002). For the evaluation purpose, the authors generated benchmark problems using expected time to compute model proposed in Braun et al. (2001). The experiment results show that the proposed algorithm achieved the best makespan values in all instances. Moreover, the proposed algorithm has the lowest standard

deviation. It is clear from the results that the proposed PSO algorithm was able to achieve good results. However, the experiments were conducted on a static job scheduling scenario which is not enough to make a conclusion about the algorithm efficiency. Hence, more studies and experiments required using dynamic job scheduling scenario in order to understand the algorithm behaviour.

Another study which implemented PSO approach has been proposed by Izakian, Ladani, Zamanifar, and Abraham (2009). The study objectives are to minimize the makespan as well as flowtime simultaneously. The approach's implementation was based on static environment using expected time to compute model to estimate the required time for processing task in a machine. The proposed algorithm was compared with Fuzzy PSO presented in Abraham et al. (2006). The experiment results show that the proposed PSO approach performs better than Fuzzy PSO. However, no details were provided regarding the benchmark problem and the flowtime results were not reported in the study. In addition, the proposed PSO algorithm has not been implemented with dynamic environment. Therefore, the proposed approach requires more experiments with dynamic environment and compared with other metaheuristics algorithms as well.

Another study using PSO to schedule jobs in heterogeneous computing systems has been proposed by Izakian, Abraham, and Snasel (2009b). The proposed algorithm aims to minimize the makespan value as a performance criterion. The study compared the proposed algorithm with GA presented in Braun et al. (2001) and PSO presented in Salman et al. (2002). The conducted experiment is based on static environment using expected time to compute model proposed in Braun et al. (2001). The empirical results show that the proposed PSO algorithm achieved the best makespan in all

instances. In addition, the algorithm convergence time was the lowest in most instances. In spite of these good results achieved using the proposed PSO algorithm, more experiments are required using dynamic environment in order to evaluate the algorithm robustness.

A comparison of four metaheuristics algorithms for task scheduling in computational grid system was presented by Meihong and Wenhua (2010). The algorithms used in their study for comparison are genetic algorithm, ant colony optimization algorithm, particle swarm optimization algorithm, and simulated annealing algorithms. The evaluation criteria are makespan and the mean response time. The authors conducted experiments using static environment. The results show that PSO algorithm has the best performance among the other algorithms. However, the experiments were conducted in static environment and very small scenario (5 users and 3 resources). Therefore, the robustness of the compared algorithms is not proven. In addition, only classical versions of the algorithms are used while enhanced versions are better in terms of performance. In order to obtain a clear picture about which metaheuristics is better, more investigations and experiments are required using a known benchmark such as the one presented in Braun et al. (2001).

Izakian, Ladani, Abraham, and Snasel (2010) proposed a discrete particle swarm optimization for job scheduling in grid computing. Their approach aims to minimize the makespan and flowtime simultaneously in grid computing. In their study, they provide two representations for mapping between problem solution and PSO particle. The first representation used a direct encoding that is a vector with size equal to the number of tasks. Each element in the vector represents the machine number. The second representation used a binary matrix size of (jobs number * machines number).

The matrix was represented with values either 0 or 1. The benchmark problem used to evaluate the proposed algorithm is based on expected time to compute model presented by Braun et al. (2001). The proposed algorithm was compared with GA, ACO, PSO, and Fuzzy PSO algorithms. The experiment results show that the proposed algorithm achieved good results in makespan reduction, while for flowtime, the algorithm performed the worst. Although the study aims to minimize makespan and flowtime, the contradiction is clear between them such that the algorithm could not reduce both of them simultaneously. This contradiction is mentioned by Xhafa and Abraham (2010) in grid computing as well. In general, the proposed algorithm performs better than other algorithms. However, the experiments were conducted using only static environment. Therefore, more experiments on dynamic environment are required to conclude the performance of the proposed algorithm.

Another study using fuzzy particle swarm optimization for job scheduling in grid computing has been proposed in H. Liu et al. (2010). In their algorithm, they extended the velocity and position of particles from the real vectors to fuzzy matrices. The advantages of using fuzzy matrices in PSO are the speed of convergence and the increase of the ability to find a faster and feasible solution. The study used the makespan criterion to measure the algorithm performance. The performance of the proposed algorithm was compared with genetic algorithm and simulated annealing algorithm. The experiment results show that the proposed algorithm outperforms the other algorithm especially in terms of execution time. However, the study did not use a common benchmark in order to evaluate the proposed algorithm with other approaches. In addition, only genetic algorithm and simulated annealing algorithms were used for comparison which is also not enough to give a complete picture.

Moreover, the experiments were conducted with static environment only. Therefore, it is not clear how the proposed algorithm will behave in dynamic environment.

2.2 Hybrid Approaches in Job Scheduling

The term hybrid refers to the concept of combining two or more algorithms in order to complement each other hoping to achieve a better performance. Hybridization could be between any types of algorithms such as heuristic and metaheuristics algorithms (Talbi, 2013a). There are two levels of hybridization, namely low and high levels (Xhafa, Kolodziej, Barolli, & Fundo, 2011). In low level (also called strong coupled) hybridization, the algorithms interchange their inner procedures. One of the hybrid algorithms is considered as the main while the others are subordinate algorithms. The low level hybridization could be presented as $Algorithm_1(Algorithm_2)$, where $Algorithm_1$ is the main algorithm and $Algorithm_2$ is the subordinate algorithm. On the other hand, high level hybridization could be represented as $Algorithm_1 + Algorithm_2 + \cdots + Algorithm_n$ where $Algorithm_1$ will start first, then it will call $Algorithm_2$ after it finishes its process and so on. The sequences could be repeated any number of times (depending on the algorithm design). Hybrid approach achieved very good performance in many fields compared with stand-alone approaches (Kolodziej, 2012). Hybridizing algorithms show promising results in job scheduling for grid computing.

Ritchie and Levine (2004) published a hybrid approach between ant colony optimization and tabu search algorithm for job scheduling in heterogeneous computing environment. The idea behind this hybridization is that the tabu search algorithm executed to the best-so-far solution found by the ants after some iteration which is controlled by a parameter. In other words, tabu search algorithm is not

applied to every ant solution due to the longer processing time of tabu search algorithm. The proposed algorithm adopted ACO implementation from Dorigo and Stutzle (2003) and tabu search implementation from Ritchie and Levine (2003) which is based on the approach described in Thesen (1998). The experiments were conducted using static benchmark problems based on expected time to compute model presented by Braun et al. (2001). The implemented algorithm was compared with Min-Min, GA, Min-Min+LS, Min-Min+Tabu developed by Braun et al. (2001). The experiment results show that the proposed approach outperforms other algorithms for all instances in terms of makespan. However, the study reported that the proposed algorithm took 3.5 hours to finish the execution of 1000 iterations of ACO which is considered a very long time compared to 90 seconds according to Xhafa, Duran, et al. (2008). In addition, the experiments were conducted using only static environment and makespan criterion which is not enough to conclude the algorithm performance. Nevertheless, the study provides practical hybridization details which could be re-implemented with enhancement.

A hybrid approach for job scheduling in computational grid systems proposed by Xhafa (2007). The proposed algorithm is based on memetic algorithms and several local search algorithms. The idea of the hybridization is that MA can use any of the 16 local search algorithms during the search process. In addition, MA algorithm hybridized with TS algorithm as a high level hybridization (MA+TS). The proposed algorithm aims to minimize the makespan and flowtime as a multi-objective optimization. The study investigated the proposed algorithms on static and dynamic grid computing environment. For static environment, the evaluation is based on ETC model proposed in Braun et al. (2001) and the proposed algorithms MA and MA+TS compared with two versions of GA implemented in Braun et al. (2001) and Carretero

and Xhafa (2006). Furthermore, the study conducted experiments on dynamic grid computing environment using HyperSim simulator developed by Phatanapherom et al. (2003). The experiment results show that for static environment, MA+TS outperforms all other algorithms in terms of makespan, while MA outperforms all other algorithms in terms of flowtime. Similar results were also obtained for dynamic environment experiments. However, the experiments on dynamic environment did not evaluate the proposed algorithms with other algorithms (only comparing MA with MA+TS). Therefore, more evaluation is needed with other algorithms especially in dynamic environment. Nevertheless, the study provides a good foundation regarding the performance of local search with metaheuristics.

Another study called "An Improved Ant Colony Algorithm for Grid Scheduling Problem" was conducted by Bagherzadeh and MadadyarAdeh (2009) to improve AS algorithm. They argued that using the traditional AS (task, machine) will not achieve the optimal solution. In their approach, they hybridized between MaxStd and AS methods. The idea behind that is to give a higher probability to tasks that have higher standard deviation. They conducted experiments with twelve different types of problems. The results show an improvement in makespan and utilization vary from 3% to 29% depending on the problem type. However, the proposed algorithm was compared with ant system algorithm, and not ant colony system, which is considered the newest version in ant colony optimization algorithms.

A study on low level hybridization between Genetic Algorithm and Tabu Search GA(TS) for job scheduling in grid computing has been published by Xhafa, Gonzalez, Dahal, and Abraham (2009). The idea behind this hybridization is to enforce more exploitation of solution space using the smart process of tabu search algorithm. The

proposed approach starts with genetic algorithm as the main algorithm and calls tabu search algorithm to improve the population individuals. The hybrid algorithm considers the scheduling problem as a bi-objective optimization problem. The study aims to minimize makespan as a primary objective and then minimize flowtime as a secondary objective. The implementation and comparison of genetic algorithm and tabu search is adopted from Carretero et al. (2007) and Xhafa, Carretero, et al. (2009) respectively. For evaluation purpose, the experiments were conducted using a grid computing simulator developed by Xhafa et al. (2007). A three static scenario was generated using expected time to compute model as a benchmark problem for experiments. In terms of makespan, the experiment results show that the proposed hybrid algorithm performs better than GA and TS for small and medium size instances. However, GA(TS) achieved the worst value for large size problems. In term of flowtime, GA(TS) achieved the best result for large size instances and the worse values for other instances. The authors concluded that the proposed hybrid algorithm outperforms other algorithms in terms of makespan for small and medium size grid scenarios which is the prime objective of the study. In addition, the experiment results show that the bi-objective optimization problem in grid computing is a contradictive problem. However, the experiments are very limited in terms of static instances and the algorithms compared with. In addition, it is very important to test any hybrid approach on dynamic grid computing simulator in order to observe how the algorithm can cope with dynamic change.

A hybrid genetic algorithm based scheduler with secure and task abortion features has been proposed by Kolodziej, Xhafa, and Bogdanski (2010). The study proposed four hybrid genetic algorithms, namely Secure Genetic Algorithm (SGA), Risky Genetic Algorithm (RGA), Player's Genetic Algorithm (PGA1), and Player's Minimum

Completion Time (PMCT). Besides the proposed algorithm for scheduling jobs in grid computing, the authors added security and task abortion mechanisms which are considered as crucial issues in grid systems. The performance of the proposed algorithm was measured through the makespan and flowtime metrics as bi-level optimization problem. The study focused on independent job scheduling problem in batch mode as described in Xhafa and Abraham (2010). In addition, the problem was formulated using expected time to compute matrix model presented by Ali et al. (2000b). The implementation of the proposed genetic algorithm for independent batch scheduling was adopted from Carretero et al. (2007). The experiments were conducted using discrete event-based grid simulator called HyperSim-G developed by Xhafa and Carretero (2009). Using HyperSim-G, static and dynamic sets of instances were generated as benchmark problems in order to evaluate the proposed algorithm. The experiment results show that PMCT algorithm outperforms other algorithms for static and dynamic instances in terms of makespan. For the case of flowtime, all results are similar except for very large grid size where PMCT did not perform well as other algorithms. Despite the fact that security and task abortion are very important features in grid computing, security could be separated from the scheduler layer to improve the algorithm performance. Regarding task abortion, more studies are required to define a sufficient mechanism for such a complex process.

Song, Sun, and Cao (2010) presented a study on the convergence of converse ant colony algorithm for job shop scheduling problem. In their study, they addressed two problems with traditional ACO algorithm, slow and easy to fall into the local optimal solution. To solve these problems, they proposed an algorithm called “Hybrid Converse Ant Colony Optimization (HCACO)” with global convergence. HCACO algorithm uses ACO algorithm with simulated annealing algorithm. The hybrid

algorithm can quickly rule out the poor solutions, so that the pheromone of optimal path will be updated immediately, and the search time will be reduced. Because of using SA algorithm which has the probability of escaping from the local optimization, it is sure that ants will not fall into local optimal solution. The experiment was conducted using a simulator with 13 hard benchmark problems. The results presented a comparison between HCACO, Parallel Genetic Algorithm (PGA2), and ACO. From the results, HCACO shows the best result in terms of average relative error percentage which is smaller than PGA and ACO. The calculation time of HCACO and PGA was equal.

A paper written by Wang, Duan, Jiang, and Zhu (2010) presented a new algorithm for grid task scheduling using Genetic and Simulated Annealing algorithms (GSA). The algorithm combined genetic algorithm with simulated annealing algorithm. They pointed out that in spite of GA being fast in searching rate at the beginning; it suffers from trapping in local minimum, while SA takes a long time to get the global minimum. Based on those reasons, the authors combined GA with SA to inherent the convergence property of simulated annealing and parallelism capability of genetic algorithm. The hybrid algorithm GSA will start with GA which is stopped prematurely after satisfying the termination condition. After that, each node visited at the last generation with the best node found overall are taken as starting input for the simulated annealing algorithm. The experiment results show that GSA performs better than GA and SA. GSA has the ability to converge to a global minimum because of SA property. However, the experiment was conducted using only 15 tasks and 3 resources which are not enough to prove the robustness of the algorithm and the feasibility of calculation time for big problem space such as 100 tasks and more than 10 resources. Therefore, more investigation is needed to evaluate the hybrid algorithm GSA.

A hybrid approach between population and local search algorithms has been developed by Xhafa, Kolodziej, Barolli, and Fundo (2011). The study represents a high level hybridization between GA and TS, named GA+TS algorithm for scheduling in grid computing. The algorithm starts with GA for a specific time, and then passes the results to TS algorithm as an initial solution. TS algorithm will search the neighbourhood of the initial solution until the termination condition is met. The authors expected that GA will explore the solution space widely, while TS will make in-depth exploration of the best solution found by GA. The objective of the proposed algorithm is to enhance the makespan as a primary objective and flowtime as a complement objective. This type of optimization scheme is referred as a hierarchic optimization with priority to makespan criterion. Both algorithms, GA and TS, have been implemented and evaluated for job scheduling in computational grid in Carretero et al. (2007) and Xhafa, Carretero, et al. (2009). The experiments were conducted using a HyperSim-G simulator developed by Xhafa et al. (2007). The study considered static and dynamic grid computing environment. The proposed algorithm evaluated with GA, TS and GA(TS) algorithms proposed in Carretero et al. (2007), Xhafa, Carretero, et al. (2009), and Xhafa, Gonzalez, et al. (2009). The experiment results show that the proposed algorithm outperforms other algorithms only in one instance for static environment in terms of makespan and flowtime. While for dynamic environment, the proposed algorithm achieved good results only in one small size grid instance in terms of makespan. However, the GA+TS did not perform good compared to other algorithms in terms of flowtime. In spite of these results, the concept of using local search with population algorithm needs more investigation specifically, the hybridization of different level and different population algorithms.

A study conducted by Xhafa, Kolodziej, Barolli, Kolici, et al. (2011) proposed a hybrid approach between GA and TS for independent batch job scheduling in grid computing. The hybrid algorithm aims to optimize the makespan and flowtime as a bio-objectives scheduling problem. In addition, the authors proposed hierarchic and simultaneous approaches for optimizing makespan and flowtime as well. Two types of hybridization provided are low and high level hybridizations, which in turn result in GA(TS) and GA+TS algorithms. The experiments were conducted considering static and dynamic grid computing environment using HyperSim-G simulator developed by Xhafa et al. (2007). The proposed algorithms were compared with GA proposed in Carretero et al. (2007) and TS represented in Xhafa, Carretero, et al. (2009). The experiment results show that the proposed hybrid algorithms outperform the other stand-alone algorithms in makespan criterion. However, in terms of flowtime criterion, GA and TS stand-alone algorithms outperform the proposed hybrid algorithm; such a contradiction is normal for job scheduling in grid computing. In spite of the limitation on the experiments and benchmark problem, the study illustrated the implementation of the hybrid algorithms clearly.

A study has been proposed by Xhafa, Duran, and Kolodziej (2011) on exploitation and exploration of solution space for job scheduling in computational grid systems. The study aimed to utilize the population-based algorithm as an exploration mechanism for search space and hybridize it with local search as an exploitation mechanism. The authors proposed Memetic Algorithms (MAs) as a population-based exploration method and Hill Climbing (HC) and Tabu Search (TS) algorithms as exploitations methods. The proposed algorithm was evaluated using static benchmark problem adopted from Braun et al. (2001) and dynamic benchmark problem generated using HyperSim simulator developed by Phatanapherom et al. (2003). The proposed

hybrid MA+TS was compared with GA results implemented in Braun et al. (2001). The experiment results show that the proposed algorithm achieved the best makespan values for all instances. However, for flowtime results, the non-hybrid approach, that is MA approach, achieved the best flowtime values. Similar results were obtained for dynamic experiments where MA+TS achieved the best results for makespan, but worst for flowtime. It is clear from the results that the proposed algorithm faced contradictory optimization problem which is not possible to improve both objectives of makespan and flowtime.

Nithya and Shanmugam (2011) proposed new Hybrid Ant Colony Optimization (HACO) algorithm for job scheduling in grid computing. In their research, they focused on high performance computing criteria to decrease the execution time in grid computing. The proposed algorithm is based on ant colony optimization for dynamic batch mode heuristic mapping. The new approach considers each job as an ant in the colony and the pheromone details are provided to help in finding the optimal solution. The proposed algorithm uses a new rule for updating the pheromone, and probability matrix calculation formula in order to increase the efficiency of the existing ant colony algorithm. Different types of experiments were conducted. The results show that the proposed algorithm reduces the makespan in reasonable time. However, the load balancing criterion is neglected in their research which is a very important factor in grid computing performance and throughput.

Another study presented by Wei, Zhang, Li, and Li (2012) aimed to improve the ant colony algorithm for grid task scheduling. They introduced a new type of pheromone and a new node redistribution rule, at the same time, the algorithm can track the performance of resources and tag it. The proposed algorithm considers the load

balance, task execution time, and resource fault. The approach replaces the path pheromone into node pheromone to describe the handling capacity of current resource. The meaning of pheromone in this algorithm is resource processing capacity. The proposed system model consists of task receiver, task scheduler, and resource information service collector. The new pheromone is called resource suitability. By using this formula, the algorithm can evaluate the resource stability and increase its pheromone. Another important improvement in this algorithm is the resource redistribution rule which handles the unsuccessful task processing. The results are compared with traditional ant algorithm. They claimed that their algorithm performs better than the basic ant algorithm. The idea of a second type of pheromone is interesting, but the experiment is very limited. In order to prove this concept, more experiments are needed and more comparison with other algorithms such as ant colony systems, and ant systems are required.

A recent study on hybrid approach between ant colony optimization and cuckoo search algorithm for job scheduling in grid computing has been presented by Raju, Babukarthik, and Dhavachelvan (2013). The authors tried to combine the advantages of pheromone in ant colony optimization with local search feature of cuckoo search algorithm. The study aimed to minimize the makespan value for job scheduling in computational grid systems. The experiments were simulated using parallel computing toolbox in MATLAB. The proposed algorithm performance was compared with ant colony optimization algorithm using static scenario developed by the authors. The experiment results show that the proposed algorithm executes faster than ACO algorithm. However, the makespan values which is supposed to be the study objective is not reported. In addition, the paper did not specify which ant colony optimization member was used in the hybridization. Moreover, the experiments were very limited

in terms of tasks and resources used and the benchmarks were unknown. Therefore, more studies are required to investigate the algorithm performance, especially in dynamic environment. Nevertheless, the idea of hybridizing new metaheuristics algorithms such as cuckoo search algorithm with ant colony optimization is a useful idea.

2.3 Grid Simulator

One of the main factors that affect the performance of a grid computing is the workload to which the system is subjected (Feitelson, 2013). Evaluating the job scheduling algorithm with wrong workloads will lead to erroneous results which cannot be relied upon (Smith, 2007). The workload could be classified into static and dynamic workload. The author also stated that the differences between static and dynamic workloads may have subtle implications for performance evaluation. Therefore, an experiment that utilizes static workloads is incapable to evaluate the performance of job scheduling algorithm. Feitelson (2013) has also stated that static workload cannot be considered as valid samples of real dynamic workloads. Organizations such as System Performance Evaluation Consortium, Grid Workloads Archive and Transaction Processing Performance Council provide several benchmarks on CPU, network file system, web servers, cluster, grid, database and parallel distributed systems for evaluation of computer systems (Feitelson, 2013). These benchmarks are useful to be analyzed and modelled.

One of the successful models for heterogeneous static computing system is Expected Time to Compute (ETC) proposed by Ali et al. (2000). The model arranges the information in a two dimension matrix called ETC matrix. Each entry in the matrix, ETC $[_{i, j}]$, represents the expected execution time of task i on machine j. In ETC

matrix, the elements along a row represent the estimates of the expected execution times of a given task on different machines, while the elements along a column give the estimates of the expected times of different tasks on a given machine. Two methods are used in ETC model, namely Range Based ETC Matrix Generation and Coefficient-of-Variation Based ETC Matrix Generation. The first method used normal distribution while the second method used gamma distribution. However, there is a limitation with the ETC model. Its computing capacity of resources remains unchanged (static) during tasks execution. Thus, ETC model does not reflect the real dynamic environment in grid computing (Xhafa & Abraham, 2008a).

GridSim is one of the popular simulators for static job scheduling in grid computing (Hao, Liu, & Wen, 2012). GridSim is a java-based discrete-event simulation toolkit which can simulate heterogeneous resources, users, applications, brokers and schedulers in grid computing. However, GridSim suffers when simulating more than 2,000 grid sites concurrently due to the memory consumption. In addition, GridSim does not simulate the failure of resources which is one of the dynamic natures in real grid computing environment. A simulator for mapping jobs to resources in grid environment was also proposed by Chaturvedi and Sahu (2011). They developed their simulator using C++ language for ten metaheuristics algorithms. However, their simulator is based on ETC model which can simulate only static environment.

Caron, Garonne, and Tsaregorodtsev (2007) developed a simulator for many clusters of heterogeneous nodes belonging to a local network. Their simulator was developed based on Simgrid toolkit proposed by Grosan and Abraham (2007). They used the improved Simgrid simulator in their experiments for batch system. Probability distributions such as Gamma, Gaussian and Poisson were available to simulate the

pattern of arrival of jobs. However, the simulator lacks the attribute of resource failure.

For large distributed grid systems and complex job scheduling, a simulator called GangSim was developed by Dumitrescu and Foster (2005). The simulator focuses on the interactions between local and community reservation allocation policies. GangSim allows parallel execution running on real resources which make scheduling process faster. GangSim includes components such as external scheduler, local scheduler, data scheduler, monitoring distribution points, and virtual organization. In spite of all these components in GangSim, the authors stated that GangSim is still far from an accurate simulation of the grid environment, primarily due to various idiosyncratic features.

Grid World Archive (GWA) provides archives of operational data that can be used in evaluating job scheduling algorithms. However, GWA archives lack details and systematic description of the grid or cluster resources and from where the data were collected (Klusacek & Rudova, 2010). In addition, information on the background of the load, resource failures or specific user's requests were not provided. It can be seen that present models cannot fully simulate the dynamic nature of jobs and resources in the grid environment.

2.4 Conceptual Framework

This study focused on metaheuristics algorithms domain specifically on the hybridization between ant colony system, genetic algorithm, and tabu search algorithms. Figure 2.7 illustrates the research conceptual framework.

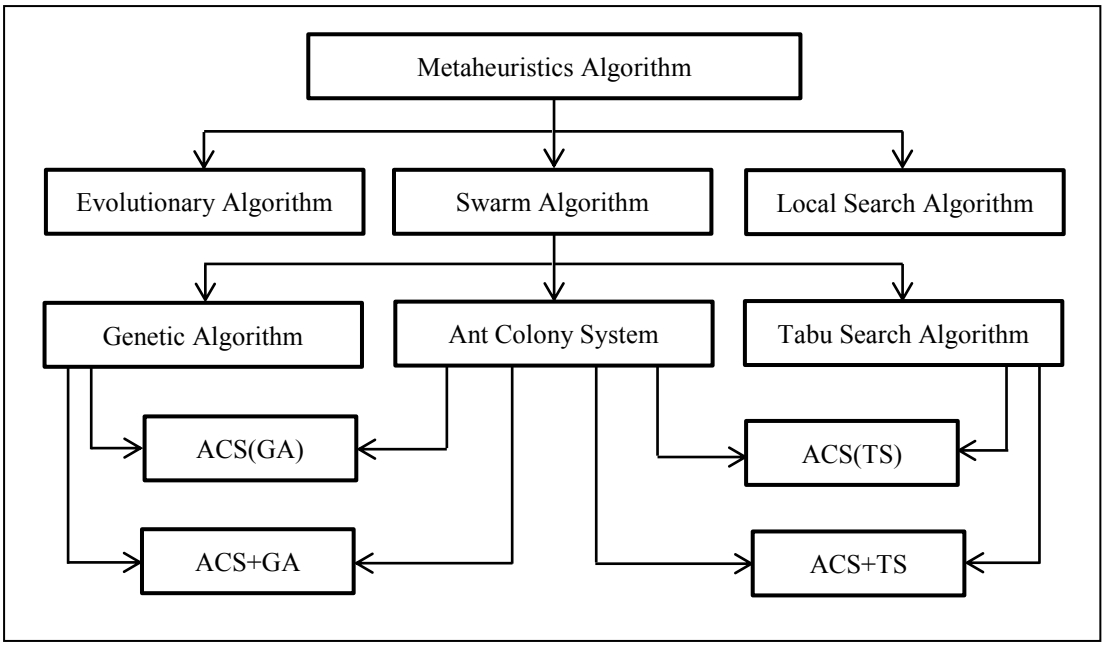

Figure 2.7. Research conceptual framework

In Figure 2.7, the framework starts with metaheuristics algorithms components. There are several categorise of metaheuristics which could be classified as population or individual based, swarm intelligence, local search, nature inspired algorithm. However, there is no standard classification for metaheuristics algorithms (Blum & Roli, 2003). This study selects ant colony system algorithm from swarm intelligence branch, genetic algorithm from evolutionary branch, and tabu search from local search branch. From these three algorithms, four hybridized algorithms are proposed in this study, namely ACS(GA), ACS+GA, ACS(TS), and ACS+TS. These hybrid algorithms are based on low and high level of hybridization between ACS, GA, and TS algorithms. The low level hybridization will enhance the exploration mechanism of ACS algorithm, while the high level hybridization will refine the final solution found by ACS algorithm. For evaluation purpose, the proposed hybrid algorithms are implemented and evaluated on job scheduling problem in static and dynamic grid computing system.

2.5 Summary

It is clear from the previous studies that scheduling problem is NP-complete problem. So far, there is no exact algorithm for solving NP-complete problems. Therefore, approximate algorithms were used for solving such a problem. All the approximate algorithms do not guarantee to find the optimum solution but they try to reach a near optimum solution within reasonable time and resources. ACS is one of these algorithms which have shown a good performance in solving different types of optimization problems. However, in huge instances problem, ACS suffers from stagnation problem which makes it insufficient in terms of computation time and solution quality. The huge instance means the search space is very big. In order to search a wide area of that search space, ants in ACS algorithm need to explore more nodes and arcs. In addition, the number of ants needs to be increased. As the mechanism of exploration in ACS influenced by random selection, any wrong path selection in terms of cost will affect the whole solution quality. Therefore, by increasing the exploration rate, the rate of wrong selection will also be increased and the final solution definitely will be out of quality (it was clear in most of the literature review that the rate of exploration is 0.1). On the other hand, increasing the number of ants will make the search process very slow because each ant will construct its own solution. Therefore, increasing any parameter value related to those issues will not give a better result, instead of that; it will give a worse result. Hence, ACS needs better exploration mechanism which is not based on random selection. Tabu search algorithm shows very fast convergence with reasonable processing time. Genetic algorithm and tabu search are considered as good candidates to do the enhancement process for the solution found by ACS algorithm. Therefore, this study proposes

hybrid ACS with GA and TS algorithms. The hybrid approaches are implemented in low and high level of hybridization.

CHAPTER THREE

RESEARCH METHODOLOGY

This chapter presents the framework and methodology of this study to implement the hybrid ACS with GA and TS algorithms. In addition, the level of hybridization for each approach is discussed in this chapter as well. Moreover, this chapter provides the evaluation details.

The rest of this chapter is organised as follows. Section 3.1 discusses the research framework and Section 3.2 describes the methodology, techniques and the proposed algorithms. The summary is presented in Section 3.3.

3.1 Research Framework

The research framework started with enhancing the exploration mechanism in ACS algorithm by implementing low level hybridization with GA and TS algorithms as shows in Figure 3.1. Then, the GA and TS algorithms are hybridized with ACS algorithm in a high level order to refine the solution found by ACS algorithm. The design and development of the grid computing simulator is undertaken in the third phase. Finally, phase four focuses on the evaluation of the proposed algorithms. The following sections describe the methods and techniques used in each phase of the research framework.

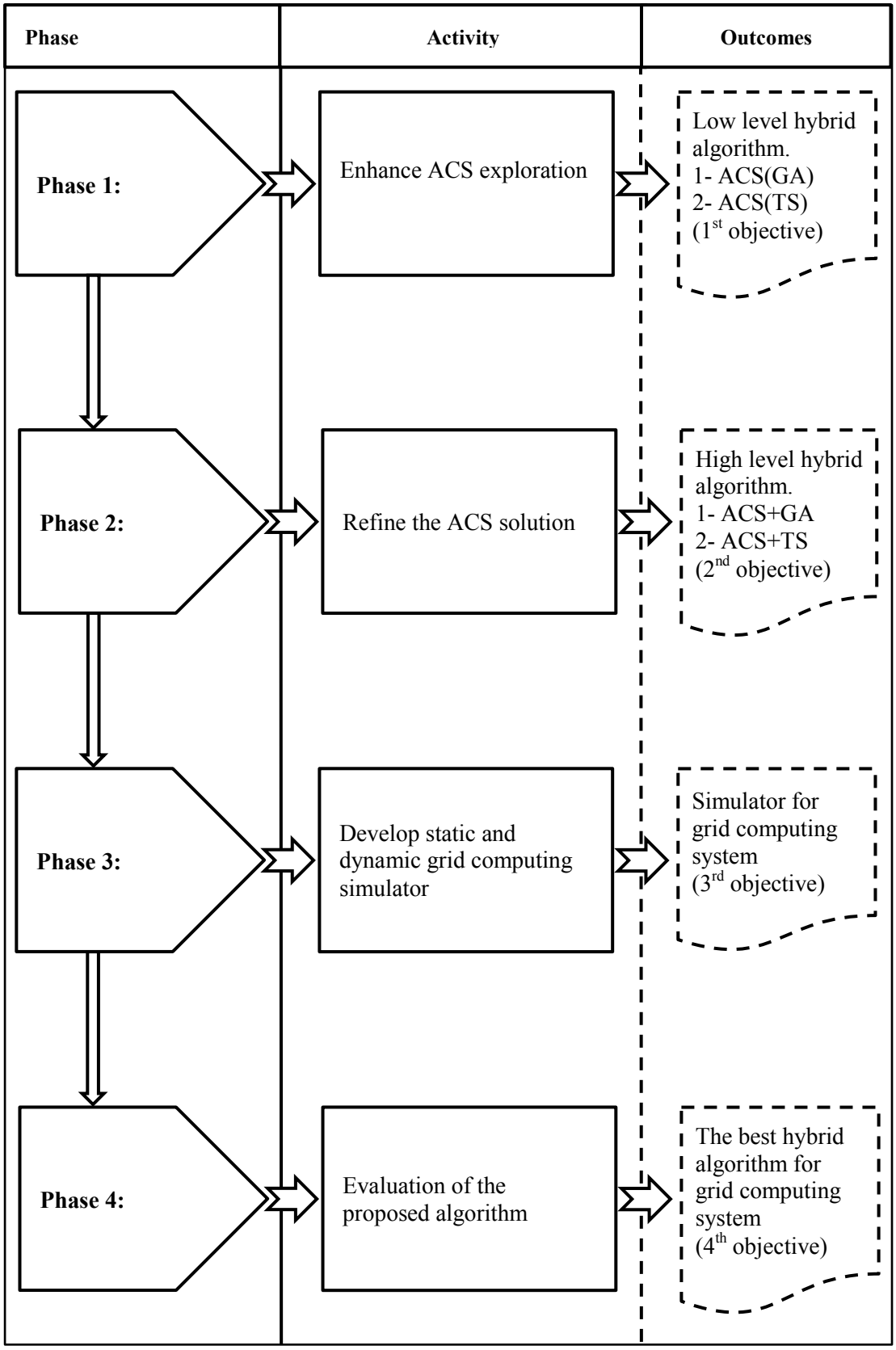

Figure 3.1. The Research Framework

3.2 Research Methodology

The methods that have been used in conducting the research are explained in the following sections.

3.2.1 Problem Formulation

Job scheduling problem consists of complex job involving the execution of multiple tasks. Each job contains one or more tasks (Kolodziej, 2012). This study considered a static and dynamic grid computing system based on batch mode. In batch mode, the tasks are grouped into a batches and each batch assigned to the resources via the scheduler. In addition, this study deals with independent tasks that are tasks with no relation between each other. The task size is expressed using Million of Instruction (MI) and the resource capacity expressed using Million of Instruction Per Second (MIPS) (Kolodziej, 2012). The time required to process a task on a resource is calculated using Expected Time to Compute (ETC) model proposed by Braun et al. (2001) as follows:

$$ETC[i,j] = \frac{task_i}{machine_j} \tag{3.1}$$

$ETC_{n\times m}$ is a matrix with two dimensions $n \times m$ where n is the number of tasks and m is the number of machines. In addition, each machine has a load to process before processing the new tasks. The previous load expressed using ready time vector (Kolodziej, 2012). The ready time vector of all machines is defined as:

$$ready\_time = [ready_1, ready_2, \dots, ready_m]$$

The completion time of $machine_j$ is calculated using:

$$completion[j] = ready_j + \sum_{i \in Task(j)} ETC[i,j], \tag{3.2}$$

Where $Task(j)$ is the set of tasks assigned to the $machine\ j$ (Kolodziej, 2012).

The $completion[j]$ parameters are the coordinates of the following completion vector:

$$completion = \left[completion[1], completion[2], \dots, completion[m]\right]^T \tag{3.3}$$

Using completion vector, the makespan calculated using the following equation:

$$makespan = \max_{j \in M} (\text{completion[j]}) \tag{3.4}$$

where M is the number of machines (Kolodziej, 2012).

The workflow of the sequence of tasks on a given $machine\ i$ is calculated using the following equation:

$$WF[j] = ready_j + \sum_{i \in Sorted[j]} ETC[i,j] \tag{3.5}$$

Where $WF[j]$ is the workflow of the $machine\ j$, $Sorted[j]$ is a set of tasks assigned to the $machine\ j$ sorted in ascending order by the corresponding ETC values (Kolodziej, 2012).

The flowtime value is the sum of $WF[j]$ parameters using the following equation:

$$Flowtime = \sum_{j \in M} WF[j] \tag{3.6}$$

The utilization metric is calculated using the following equation (Kolodziej, 2012):

$$average\ utilization = \frac{\sum_{\{j \in Machines\}} completion[j]}{makespan \cdot numner\ of\ machines} \tag{3.7}$$

3.2.2 Dynamic Expected Time to Compute

For ACS implementation, the heuristic information needs to be defined. For static environment, heuristic value is calculated from the Expected Time to Complete (ETC) matrix using $\{1 / (\mathrm{ETC}_{ij} + \mathrm{Load}_j)\}$ where ETC_{ij} represents the expected time to compute task i on machine j Equation (3.1), and Load_j is the previous load assigned to machine j (Ku-Mahamud & Alobaedy, 2012). For dynamic environment, this study will calculate the heuristic value from the Dynamic Expected Time to Complete (DETC) matrix using $\{1 / (\mathrm{DETC}_{ij} + \mathrm{Load}_j)\}$ where DETC_{ij} represents the dynamic expected time to compute task i on machine j, and Load_j is the previous load assigned to machine j. Longer computing time and more loads will produce a smaller heuristic value, which will make the probability of selecting this machine smaller and vice versa.

3.2.3 Solution Encoding

One of the most important parts of any metaheuristics algorithms is how to encode the solution which is related to the algorithm data structure (Michalewicz, 1999). This study implemented the solution encoding using a vector (Kołodziej & Khan, 2012). The size of the vector is equal to the number of tasks. The vector value indicates the

machine number; therefore, the vector values are in the range of (1 – number of machines) (Xhafa et al., 2011). For example:

Solution_Vector [1] = 2, means task 1 is assigned to machine number 2.

This representation implemented for ant colony system, tabu search, and genetic algorithm. For ant colony system, each ant holds an empty vector which represents the schedule solution. During construction phase, each ant will assign a value to the vector which is a machine number. Once the construction phase is finished, the ant has the complete vector which is a complete schedule solution. Figure 3.2 shows the solution vector used by the ants.

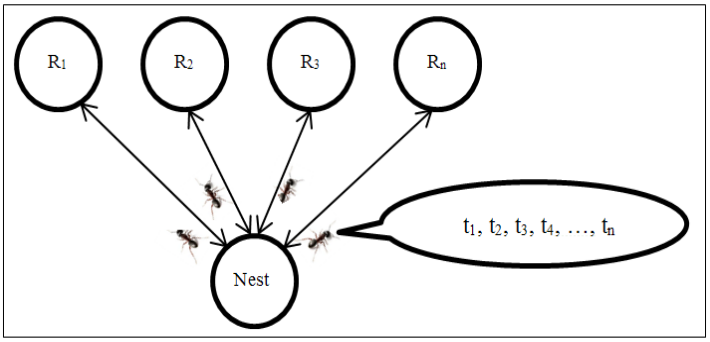

Figure 3.2. The solution vector used by the ants

For tabu search algorithm, same representation implemented as a solution trajectory. TS algorithm searched the vector neighbourhood using methods, such as swapping the adjacent machine. The vector will always represent a valid schedule solution as long as the vector values in the range of (1- number of machines).

For genetic algorithm, the solution vector considered as a chromosome. The following scenario is used in demonstrating the practicality of the proposed representation. A

scheduler has to assign five tasks (t) to four resources (r). Figure 3.3 shows the solution vectors contain the resource number.

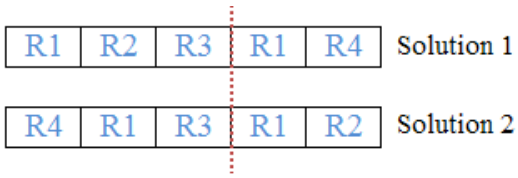

Figure 3.3. Solution vectors used by genetic algorithm

Note that the tasks sequences are fixed in ascending order ($t_1, t_2, \ldots, t_n$). This type of solution vector contains less information (only resource number) and easier to manipulate with operations such as crossover in genetic algorithm. For example, applying crossover after the third gene in Figure 3.3 will produce new solutions as shows in Figure 3.4.

Figure 3.4. The new solution vectors produced by crossover operator

In validating the solution, changing the resource order will always produce valid solution even if there are resources that are assigned more than once or not utilized at all. In other words, validation of the solution can be omitted. Thus applying this type of representation will reduce the calculation process and time.

3.2.4 Objective Function

There are many criteria in job scheduling to measure the solution quality, such as makespan, flowtime, utilization, matching, and balance. Due to the importance of

makespan metric, this is study considered the makespan value as the main objective to minimize using the following fitness function for GA and ACS (Braun et al., 2001):

$$Fitness = \frac{1}{makespan} \tag{3.8}$$

Makespan value calculated using equation (3.4). Solution with smaller makespan value means it has higher fitness value. For TS algorithm, the makespan value is the fitness value itself, that is: $Fitness = makepsan$.

3.2.5 Ant Colony System Algorithm Implementation

ACS algorithm starts with initializing the parameters and pheromone trails. The initial pheromone is calculated as (Dorigo & Stutzle, 2004):

$$1/((nearest - neighbor - solutin) \cdot (number\ of\ machine \cdot number\ of\ tasks)) \tag{3.9}$$

Once the initializing process is done, each ant in ACS starts solution construction process using the following equations:

$$P_{ij}^{Antk} = \begin{cases} argmax\ \left\{[t_{ij}] \,.\, [1\ /\ (DETC_{ij} + Load_j)]^{\beta}\right\}, & if\ q\ \leq q_0; \\ J & otherwise; \end{cases} \tag{3.10}$$

where $[t_{ij}]$ is the pheromone value between the task i and machine j, $DETC_{ij}$ is the dynamic expected time to compute task i on machine j (ETC_{ij} is used for static), Load_j is the previous load on machine j, β is the parameter to control the influence of the heuristic information, q is a random variable uniformly distributed in [0, 1], q_0 (0

$\leq q_0 \leq 1$) is a parameter, and J is a random variable selected according to the probability distribution using the following equation:

$$P_{ij}^{Antk} = \frac{[t_{ij}] \,.\, [1 \,/\, (DETC_{ij} + Load_j)]^{\beta}}{\sum_{j=1}^{M}[t_{ij}] \,.\, [1 \,/\, (DETC_{ij} + Load_j)]^{\beta}} \tag{3.11}$$

During the construction process, each ant will apply a local pheromone update using the following equation (Dorigo & Stutzle, 2004):

$$t_{ij} = (1 - \rho) \cdot t_{ij} + \rho \cdot t_0 \tag{3.12}$$

Where ρ ($0 < \rho < 1$) is the evaporation rate and t_0 is the initial pheromone calculated using equation (3.9).

Once all the ants finished their construction phase, the global pheromone update starts using the following equation (Dorigo & Stutzle, 2004):

$$t_{ij} \leftarrow (1 - \rho) \cdot t_{ij} + \left(\rho \cdot \Delta t_{ij}^{bij}\right), \ \forall (i,j) \in T^{bs}, \tag{3.13}$$

Where Δt_{ij}^{bij} is the fitness value found be the best-so-far ant using equation (3.8) and T^{bs} are the arcs of the best solution (Dorigo & Stutzle, 2004).

Appendix A provides the C# code for ant colony system algorithm.

3.4.6 Genetic Algorithm Implementation

Genetic algorithm consists of several methods, namely generate population, evaluation, selection, crossover, mutation, and combination operators. In this study,

each operator is implemented according to Xhafa et al. (2007a). The following are the implementation details.

I. Population

A genetic algorithm is based on a population approach which is using many chromosomes in order to apply operators, such as crossover and mutation operators. Each chromosome is representing a complete solution. For job scheduling problem in grid computing, the chromosome is a vector of integer numbers [1 - machines numbers]. The size of the vector is equal to the number of tasks and the vector index represents the task number (Xhafa et al., 2007a). For example Figure 3.5 shows two chromosomes for 5 tasks and three machines.

Chromosome[1] =	1	2	2	1	2
Chromosome[2] =	2	2	2	1	1

Figure 3.5. Chromosomes for five tasks and three machines

Each chromosome in the population is created randomly. However, for the proposed hybrid approach ACS(GA) and ACS+GA, one of the population chromosome is the best solution passed from ACS to GA.

II. Evaluation

Every chromosome's fitness value in the population is evaluated using the objective function based on makespan value as defined in equation (3.8).

III. Selection

This study implemented the *K*-tournament method as a selection operator. In *K*-tournament operator, *K* chromosomes are randomly selected from the population. From these *K* chromosomes, two chromosomes are selected with the highest fitness values. This type of selection will give chance to all individual to compete fairly (Xhafa et al., 2007a).

IV. Crossover

There are many types of crossover operators, such as one-point, two-points, multi-points, and uniform crossover. This study implemented a crossover operator knows as Fitness-Based crossover (Xhafa et al., 2007a). In Fitness-Based crossover, the crossing is made based on the fitness of the parents chromosomes (solutions). Let f_1 be the fitness of the first chromosome and f_2 is the fitness of the second chromosome. Then the probability of interchanging for each gene (machine) is calculated using:

$$p = f_1 / (f_1 + f_2) \tag{3.14}$$

In this method, if there is a large difference in the fitness values between two parent chromosomes, then it is quite probable that a chromosome of new structure will be obtained (Xhafa et al., 2007a).

V. Mutation

In this study, the mutation operator implemented based one the Re-balance mutation method (Xhafa et al., 2007a). Re-balance mutation tries to reduce the lode of the most overloaded machine by swapping jobs from the overloaded machine. The Re-balance mutation is done in two steps:

a. Choose a machine m from most overloaded machines.

b. Identify job t assigned to m and t' assigned to another machine m' such that $ETC[t'][m'] < ETC[t][m]$. Jobs t and t' are swapped.

VI. Replacement

The study implemented the replacement operator based on Steady-State Genetic Algorithm (SSGA) strategy (Xhafa et al., 2007a). In SSGA, the worse portion of the population will be replaced with the new generated chromosomes. The size of the population is maintained constantly. In spite of the risk of stagnation, using SSGA operator performs very well if a good solution is required to be find very quickly such as the case of job scheduling problem in grid computing where the time to find the solution is very restricted (Xhafa et al., 2007a).

Appendix B provides the C# code for genetic algorithm.

3.4.7 Tabu Search Algorithm Implementation

The major parts of the tabu search algorithm implementation in this study are adopted from the studies proposed by (Xhafa et al., 2009; Xhafa et al., 2011). The implemented tabu search algorithm consists of six parts as described in the following points:

I. Initial Solution

In this study, the initial solution is passed from the best solution found by the ants in ACS algorithm. Therefore, TS algorithm starts with good quality solution in order to enhance it.

II. Movements

The implemented TS algorithm applies two types of movements in order to generate new solution from the current solution's neighbourhood, namely transfer and swap. Transfer is the process of moving a job from one machine to another, while swap is the process of exchanging two jobs assigned to different machines (Xhafa et al., 2009).

III. Memory

This study implemented the recency memory that is, a tabu list with the last time each job was assigned to every machine machines (Xhafa et al., 2009). The recency memory is based on a matrix $TL_{n\times m}$where n is the number of tasks and m is the number of machines (Xhafa et al., 2011).

IV. Aspiration Criteria

Two criteria are implemented to accept the tabu movements: First, if the movement to the new solution produce better makespan value, then accept the movement. Second, if the movement to the new solution produce equal makespan to the best found solution with better flowtime value, then accept the movement.

V. Soft Diversification

Soft diversification implemented using swap load method in order to move to new search space area near to the current solution. This method starts by identifying the highest load machine and tries to swap it with the lowest load machine.

VI. Strong Diversification

During the algorithm execution, if the algorithm is not able to find better quality solution within 50 iterations, then the strong diversification is triggered. The strong diversification is based on a large perturbation of the current solution by changing the assignments of 1% of the number of jobs to the random machines (Xhafa et al., 2009).

Appendix C provides the C# code for tabu search algorithm.

3.2.8 Enhance ACS exploration

Hybridization is a term which refers to the approach that combines two or more algorithms in order to achieve a result which is not achievable using a stand-alone approach (Xhafa, Gonzalez, et al., 2009). Algorithms could be hybridized fully or partially to be able to get the best features of the combined algorithms. According to Xhafa, Kolodziej, Barolli, and Fundo (2011), there are two levels of hybridization between algorithms, namely, high level and low level which refers to the degree of coupling between the metaheuristics algorithms.

In low level hybridization, also called strongly coupled, the algorithms inter-change their inner procedures. The level of hybridization reflects the degree of inner exchange among the hybridized algorithms. In low level hybridization, one of the algorithms is the main algorithm, which calls other algorithms at any time of execution (depending on the hybridization design). The low level hybridization algorithm could be presented as $Algorithm_1(Algorithm_2)$ (Xhafa, Gonzalez, et al., 2009). In this representation, $Algorithm_1$ is the main algorithm and $Algorithm_2$ is the subordinated algorithm (Jourdan, Basseur, & Talbi, 2009; Xhafa, Kolodziej, Barolli, & Fundo, 2011).

This study has implemented both levels in order to determine the best hybridization. In low level hybridization, the combined algorithms ACS with GA "ACS(GA)" and ACS with TS "ACS(TS)" will interchange their inner procedures. ACS is the main algorithm which during its flow will call the GA and TS for enhancement. The algorithm notation ACS(GA) and ACS(TS) for low level means ACS is the main algorithm and GA or TS is the subordinated algorithm. Low level hybridization between ACS and GA will refine the solution produced by each ant in ACS. On the other hand, TS will enhance the exploration mechanism in ACS algorithm. Tabu search algorithm is based on systematic process (Glover & Laguna, 1997). Therefore, tabu search algorithm is a very suitable approach to be combined with ACS algorithm to enhance the exploration mechanism. In low level hybridization, the best solution produced by the ants is sent to the local search algorithm for enhancement. The enhanced solution is returned to the ant for pheromone update. Therefore, the ant will update the pheromone using the enhanced solution which makes the ants deposits more pheromone value. The pheromone value will influence the movements of the ants in the next iteration. Figures 3.6 and 3.7 represent the pseudocode for ACS(GA), and ACS(TS) algorithms respectively.

Procedure ACS(GA)

Step 1- Initialize the number of ants n;

Step 2- Initialize parameters and pheromone trails; Equation (3.9)

Step 3- While (Termination condition not met) Do;

Step 4- For $i = 1$ to n Do;

Step 5- Construct new solution; Equation (3.10)

Step 6- Apply local pheromone update; Equation (3.12)

Step 7- End For;

// **Genetic algorithm starts here;**

Step 8- Initialize population (*P*);

Step ***9*****-** **Add (best ant solution from ACS to *P*);**

Step 10- Evaluate (*P*); Equation (3.8)

Step 11- While (termination condition not met);

Step 12- Ṕ← Select (*P);*

Step 13 - Crossover (Ṕ);

Step 14- Mutate (Ṕ);

Step 15- Evaluate (Ṕ); Equation (3.8)

Step 16- *P* ← Replace (Ṕ ∪ P);

Step 17- End While;

// **Genetic algorithm ends here;**

Step 18- Apply Global pheromone update; Equation (3.13)

Step 19- Update best found solution s^*;

Step 20- End while;

Step 21- Return the best solution;

End Procedure;

Figure 3.6. ACS(GA) (low level) algorithm pseudocode

In ACS(GA) Figure 3.6, the first step is initializing the ants and distribute them randomly (Dorigo & Stutzle, 2004). Second step is initializing the parameters and pheromone trails. The initial pheromone is calculated using equation (3.9).

The third step is the while loop which is terminated when the condition is met, in this study the termination condition is satisfied after 90 seconds of the algorithm execution. The fourth step is the loop for each ant. The fifth step is the construction using the equation (3.10).

The sixth step is the local pheromone update using the equation (3.12). Step seven is the end of the ants loop. Step eight is the start of genetic algorithm which is start by initializing the population using random method. Step nine will add the best solution found by the ants to the population of genetic algorithm. Step ten will evaluate the population using the makespan fitness objective by equation (3.8). Step eleven is the main genetic algorithm loop using (while) with termination condition of two seconds. Step twelve is the selection process using tournament operator. Step thirteen is the crossover operator using fitness based crossover. Step fourteen is the mutation process using re-balance operator. Step fifteen will evaluate the new solution using the makespan objective function by equation (3.8). Step sixteen is the replication operator which replaces the old solution with the new generated solution using SSGA strategy. Step seventeen ends the genetic algorithm execution. The best solution found by genetic algorithm is passed back to the ant colony system algorithm.

Step eighteen is the global pheromone update using the equation (3.13). Step nineteen will update the best so far solution found by the algorithm. Step twenty ends the ant colony system algorithm and the best solution is returned by step twenty one. Appendices A and B provides the C# code for ant colony system and genetic algorithm respectively.

The proposed ACS(GA) algorithm is different than other proposed algorithm such as the one presented by Liu, Chen, Dun, Liu, and Dong (2008) which is based on using genetic algorithm to choose, cross, and mutate the parameters of ant colony algorithm. In the proposed ACS(GA) algorithm, the genetic algorithm is used to select, cross, and mutate the best-so-far solution found by ants in every cycle as illustrated in Figure 3.6 with the step "**Add (best ant solution from ACS to *P*)".** Therefore, GA

works as an exploration mechanism to explore the search space based on the solution found by the ants in ACS algorithm.

Another low level hybridization between ant colony system and tabu search algorithm is shown in Figure 3.7.

Procedure ACS(TS)

Step 1- Initialize the number of ants n;

Step 2- Initialize parameters and pheromone trails; Equation (3.9)

Step 3- While (Termination condition not met) Do;

Step 4- For i $= 1$ to n Do;

Step 5- Construct new solution; Equation (3.10)

Step 6- Apply local pheromone update; Equation (3.12)

Step 7- End For;

// **Tabu search algorithm starts here;**

Step 8- **Create solution s from best ant ACS_s^*;**

Step 9- Create global solution $s^* \leftarrow s$;

Step 10- Create tabu list TL;

Step 11- Initialize the aspiration function A;

Step 12- While (termination condition not satisfied) Do;

Step 13- Search the neighbourhood N of current solution s: $\{\hat{s} \in N(s)\}$;

Step 14- If (move from s to $\hat{s}$ is not in TL) Then;

Step 15- $s \leftarrow \hat{s}$;

Step 16- Update TL memories;

Step 17- End If;

Step 18- Else If $(f(\hat{s}) < A(f(s))$ Then;

Step 19- $s \leftarrow \hat{s}$;

Step 20- Update TL memories;

Step 21- End If;

Step 22- If $(f(s) < f(s^*))$ Then;

Step 23- $s^* = s$;

Step 24- End If;

Step 25- End While;

Step 26- $ACS\_s^* \leftarrow$ Global solution s^*;

// **Tabu search algorithm ends here;**

Step 27- Apply Global pheromone update; Equation (3.13)

Step 28- End while;

Step 29- Return Global solution ACS_s^*;

End Procedure;

Figure 3.7. ACS(TS) (low level) algorithm pseudocode

In Figure 3.7, the first step is initializing the ants and distribute them randomly (Dorigo & Stutzle, 2004). Second step is initializing the parameters and pheromone trails. The initial pheromone is calculated using equation (3.9).

The third step is the while loop which is terminated when the condition is met, in this study the termination condition is satisfied after 90 seconds of the algorithm execution. The fourth step is the loop for each ant. The fifth step is the construction using the equation (3.10). The sixth step is the local pheromone update using the equation (3.12). Step seven is the end of the ants loop.

Step eight is the starting of tabu search algorithm. TS algorithm starts by creating initial solution using the best-so-far solution found by ACS algorithm. Step nine makes a global solution which is a copy of the initial solution. Step ten initializes the tabu list to store the movement attributes. Step eleven initializes the aspiration function. Step twelve is the (while) loop which is terminated after two seconds. Step thirteen searches the neighbourhood of the current solution.

Step fourteen checks if the moving from the current solution to the neighbourhood solution is not tabu. If so, the neighbourhood solution will be saved in the current solution in step fifteen. Step sixteen will update the tabu list with old position to prevent visiting the same location. Step seventeen ends the moving condition. In case the movement to the new neighbourhood solution is tabu, the aspiration function at step eighteen will check. If the aspiration function returns true, then the neighbourhood solution will be saved in the current solution in step nineteen. Step twenty will update the tabu list with old position to prevent visiting the same location. Step twenty one ends the aspiration function. Step twenty two will check if the current solution's makespan is better than the global solution's makespan. Step twenty three saves the current solution as the global solution and ends with step twenty four.

Step twenty five ends the (while) loop and the best solution passed back to ACS algorithm in step twenty six. Ant colony system will apply global pheromone update in step twenty seven using equation (3.13). Step twenty eight ends the ACS (while) loop and the best global solution found by ACS(TS) is returned in step twenty nine.

TS algorithm works as an enhancing exploration mechanism to explore the search space based on the solution found by the ants in ACS algorithm.

Appendices A and C provides the C# code for ant colony system and tabu search algorithms respectively.

Tabu search algorithm has been implemented in several hybrid algorithms as a local search technique. A study proposed by Nagariya, Mishra, and Shrivastava (2014) implemented ACO and TS algorithms for job scheduling in computational grid. In their study, they apply TS algorithm to identify the local solution and keep the ants searching for global solution. In contrast, the proposed ACS(TS) algorithm applies TS algorithm to search globally using the best so-far-solution found by the ants at every cycle as illustrated in Figure 3.7. The task is handled by "**Create solution s from best ant ACS_s^*.**

The proposed approaches ACS(GA) and ACS(TS) show that ACS algorithm calls the local search algorithm before applying global pheromone update. The local search algorithm enhances the best-so-far solution found by the ants. After the enhancement process, ACS algorithm will apply the global pheromone update based on the solution enhanced by the local search. These hybrid approaches enhance the exploration mechanism of ACS algorithm by correcting any wrong decision taken in the previous step.

3.2.9 Refine the ACS solution

In order to enhance the solution produced by ACS algorithm, a high level hybridization is implemented in this phase. High level hybridization is also called loosely coupled hybridization, whereby each algorithm preserves its identity, in other words; each algorithm operates fully in the hybridized approach. This type of hybridization can be seen as a chain of algorithm execution ($Algorithm_1 \rightarrow Algorithm_2 \rightarrow \cdots \rightarrow Algorithm_n$) (Xhafa, Kolodziej, Barolli, & Fundo, 2011). This execution can further loop certain numbers of iterations until the termination condition is satisfied. Through the algorithms execution, the output solution is passed from $Algorithm_1$ to $Algorithm_2$ and so on.

ACS will start first to generate an initial solution for GA (in ACS+GA) and TS (in ACS+TS) algorithms. The initial solution will be passed to GA and TS for enhancement. The algorithm notation is: ACS+GA and ACS+TS for high level which means ACS starts first followed by GA or TS. In high level hybridization, each algorithm preserves its identity without any influence on each other execution. In other words, ACS algorithm execution and pheromone are total independent of GA or TS and vice versa. Figures 3.8 and 3.9 represent the high level pseudocode hybridization of ACS+GA and ACS+TS respectively.

Procedure ACS+GA

// ACS algorithm starts here;

Step 1- Initialize the number of ants n;

Step 2- Initialize parameters and pheromone trails; Equation (3.9)

Step 3- While (Termination condition not met) Do;

Step 4- For i $= 1$ to n Do;

Step 5- Construct new solution; Equation (3.10)

Step 6- Apply local pheromone update; Equation (3.12)

Step 7- End For;

Step 8- Apply Global pheromone update; Equation (3.13)

Step 9- Update best found solution s^*;

Step 10- End while;

// ACS algorithm ends here;

// Genetic algorithm starts here;

Step 11- Initialize population (*P*);

Step 12- **Add (best ant solution from ACS to *P*);**

Step 13- Evaluate (*P*); Equation (3.8)

Step 14- While (termination condition not met);

Step 15- Ṕ← Select (*P);*

Step 16 - Crossover (Ṕ);

Step 17- Mutate (Ṕ);

Step 18- Evaluate (Ṕ); Equation (3.8)

Step 19- *P* ← Replace (Ṕ ∪ P);

Step 20- End While;

// Genetic algorithm ends here;

Step 21- Return Global solution from Genetic algorithm;

End Procedure;

Figure 3.8. ACS+GA (high level) pseudocode

In ACS+GA Figure 3.8, the first step is initializing the ants and distribute them randomly (Dorigo & Stutzle, 2004). Second step is initializing the parameters and pheromone trails. The initial pheromone is calculated using equation (3.9).

The third step is the while loop which is terminated when the condition is met, in this study the termination condition is satisfied after 45 seconds of the algorithm

execution. The fourth step is the loop for each ant. The fifth step is the construction using the equation (3.10).

The sixth step is the local pheromone update using the equation (3.12). Step seven is the end of the ants loop. Step eight is the global pheromone update using the equation (3.13). Step nine will update the best-so-far solution found by the ants. Step ten ends the (while) loop of ACS algorithm and passes the best solution to genetic algorithm for refinement.

Step eleven is the start of genetic algorithm which is start by initializing the population using random method. Step twelve will add the best solution found by ACS algorithm to the population of genetic algorithm. Step thirteen will evaluate the population using the makespan fitness objective by equation (3.8). Step fourteen is the main genetic algorithm loop using (while) with termination condition of 45 seconds. Step fifteen is the selection process using tournament operator. Step sixteen is the crossover operator using fitness based crossover. Step seventeen is the mutation process using re-balance operator. Step eighteen will evaluate the new solution using the makespan objective function by equation (3.8). Step nineteen is the replication operator which replaces the old solution with the new generated solution. Step twenty ends the genetic algorithm execution. The best solution found by genetic algorithm returned by the algorithm in step twenty one.

GA algorithm works as a refinement mechanism to enhance the solution found by the ACS algorithm. Appendices A and B provides the C# code for ant colony system and genetic algorithms respectively.

High level hybridization between ACO and GA has been proposed by Kolasa and Krol (2010) for the assignment problem. In each cycle of their algorithm, the best solution of the two algorithms is selected and the search is continued by both of them. In contrast, the proposed ACS+GA algorithm applies the GA algorithm to refine the solution found by ACS algorithm. In other words, the ACS algorithm starts with a specific number of iterations or a period of time. Then the solution found by ACS algorithm is passed to GA as one of the initial chromosome population. GA will refine the solution received from ACS algorithm by applying selection, crossover, and mutation operators. The final solution will be produced by genetic algorithm as illustrated in Figure 3.8.

Another high level hybridization between ant colony system and tabu search algorithm is shown in Figure 3.9.

Procedure ACS+TS

// ACS algorithm starts here;

Step 1- Initialize the number of ants n;

Step 2- Initialize parameters and pheromone trails; Equation (3.9)

Step 3- While (Termination condition not met) Do;

Step 4- For i $= 1$ to n Do;

Step 5- Construct new solution; Equation (3.10)

Step 6- Apply local pheromone update; Equation (3.12)

Step 7- End For;

Step 8- Apply Global pheromone update; Equation (3.13)

Step 9- Update best found solution s^*;

Step 10- End while;

// ACS algorithm ends here;

// Tabu search algorithm starts here;

Step 11- **Create solution s from best ant ACS_s^*;**

Step 12- Create global solution $s^* \leftarrow s$;

Step 13- Create tabu list TL;

Step 14- Initialize the aspiration function A;

Step 15- While (termination condition not satisfied) Do;

Step 16- Search the neighbourhood N of current solution s: $\{\hat{s} \in N(s)\}$;

Step 17- If (move from s to $\hat{s}$ is not in TL) Then;

Step 18- $s \leftarrow \hat{s}$;

Step 19- Update TL memories;

Step 20- End If;

Step 21- Else If ($f(\hat{s}) < A(f(s))$ Then;

Step 22- $s \leftarrow \hat{s}$;

Step 23- Update TL memories;

Step 24- End If;

Step 25- If ($f(s) < f(s^*)$) Then;

Step 26- $s^* = s$;

Step 27- End If;

Step 28- End While;

Step 29- Return Global solution s^*;

End Procedure;

Figure 3.9. ACS+TS (high level) algorithm pseudocode

In ACS+TS Figure 3.9, the first step is initializing the ants and distribute them randomly (Dorigo & Stutzle, 2004). Second step is initializing the parameters and pheromone trails. The initial pheromone is calculated using equation (3.9).

The third step is the while loop which is terminated when the condition is met, in this study the termination condition is satisfied after 45 seconds of the algorithm execution. The fourth step is the loop for each ant. The fifth step is the construction using the equation (3.10).

The sixth step is the local pheromone update using the equation (3.12). Step seven is the end of the ants loop. Step eight is the global pheromone update using the equation (3.13). Step nine will update the best-so-far solution found by the ants. Step ten ends the (while) loop of ACS algorithm and passes the best solution to genetic algorithm for refinement.

Step eleven is the starting of tabu search algorithm. TS algorithm starts by creating initial solution using the best-so-far solution found by ACS algorithm. Step twelve makes a global solution which is a copy of the initial solution. Step thirteen initializes the tabu list to store the movement attributes. Step fourteen initializes the aspiration. Step fifteen is the (while) loop which is terminated after 54 seconds. Step sixteen searches the neighbourhood of the current solution.

Step seventeen checks if the moving from the current solution to the neighbourhood solution is not tabu. If so, the neighbourhood solution will be saved in the current solution in step eighteen. Step nineteen will update the tabu list with old position to prevent visiting the same location. Step twenty ends the moving condition. In case the movement to the new neighbourhood solution is tabu, the aspiration function at step

twenty one will be checked. If the aspiration function returns true, then the neighbourhood solution will be saved in the current solution in step twenty two. Step twenty three will update the tabu list with old position to prevent visiting the same location. The aspiration function ends at step twenty four. Step twenty five will check if the current solution better than the global solution. Step twenty six saves the current solution as the global solution and ends with step twenty seven. Step twenty eight ends the (while) loop and the best solution is returned in step twenty nine.

TS algorithm works as a refinement mechanism to enhance the solution found by the ACS algorithm.

Appendices A and C provides the C# code for ant colony system and tabu search algorithms respectively.

Tsutsui ad Fujimoto (2013) proposed ACO with TS algorithm for quadratic assignment problem. In their study, ACO and TS algorithms are run in parallel without aspiration function. While in the proposed ACS+TS algorithm, ACS executes first to find the best solution on specific number of iterations or period of time. Then, the best solution is passed from ACS to TS algorithm. The TS algorithm will refine the solution by searching the neighbourhoods of that solution as shown in Figure 3.9.

The high level hybrid approaches provide different improvement than low level hybrid approaches. The proposed ACS+GA and ACS+TS approaches show that ACS algorithm calls the local search algorithm after the algorithm execution. The solution produced by ACS algorithm pass to the local search algorithm. The local search algorithm will improve the solution as a final solution.

3.2.10 Grid Simulator Development

The simulator was developed using C# language under the windows platform. Sixteen algorithms are implemented in this simulator, namely AS, ACS, TS, GA, PSO-GELS, BABC, EBABC1, EBABC2, AS(TS), AS+TS, AS(GA), AS+GA, ACS(TS), ACS+TS, ACS(GA), and ACS+GA.

The developed simulator generates two types of grid computing environment, namely static and dynamic environments. The static environment is generated based on expected time to compute model developed by Braun et al. (2001). For dynamic environment, the developed simulator implemented dynamic expected time to compute model as proposed by this study.

The grid computing is evaluated using three metrics, namely makespan, flowtime, and utilization (Xhafa & Abraham, 2010). The simulator reports the scheduling results graphically with all experiment details.

3.2.11 Proposed Algorithm Evaluation

In order to evaluate the proposed hybrid algorithms, several experiments are conducted based on static and dynamic environments. The proposed hybrid algorithms are compared with its own stand-alone algorithm and other metaheuristics algorithms to explore the performance robustness. Some algorithms are implemented using codes adopted from literatures, while others are implemented by this study. Table 3.1 shows the algorithms that have been the implemented and the source of the algorithms.

Table 3.1

The implemented *algorithms source*

No	Algorithms	Sources
1	GA	(Passos, 2009) with some modifications.
2	TS	This study using pseudocode from Xhafa, Carretero, et al. (2008)
3	AS	(Wiener & Of, 2009) with some modifications.
4	AS(TS)	(Wiener & Of, 2009) + (Xhafa, Carretero, et al., 2008)
5	AS+TS	(Wiener & Of, 2009) + (Xhafa, Carretero, et al., 2008)
6	AS(GA)	(Wiener & Of, 2009) + (Passos, 2009)
7	AS+GA	(Wiener & Of, 2009) + (Passos, 2009)
8	ACS	(Wiener & Of, 2009) with some modifications.
9	ACS(TS)	(Wiener & Of, 2009) + (Xhafa, Carretero, et al., 2008)
10	ACS+TS	(Wiener & Of, 2009) + (Xhafa, Carretero, et al., 2008)
11	ACS(GA)	(Wiener & Of, 2009) + (Passos, 2009).
12	ACS+GA	(Wiener & Of, 2009) + (Passos, 2009).
13	PSO-GELS	(Pooranian, Shojafar, Abawajy, & Abraham, 2013)
14	BABC	This study implemented the pseudocode in Kim et al. (2013)
15	EBABC1	This study implemented the pseudocode in Kim et al. (2013)
16	EBABC2	This study implemented the pseudocode in Kim et al. (2013)

Each algorithm has executed 10 times on each benchmark problem in order to calculate the best, average and geometric mean values. As an objective function, the proposed algorithms aimed to enhance the prime criterion that is makespan value (Xhafa & Abraham, 2010). Minimizing the makespan value is considered as the most important objective which indicates the general productivity of the computational grid system (Xhafa & Abraham, 2008a). A small value of makespan indicates that the scheduler is performing well and the planning of tasks to resources is efficient. Other important performance metrics are flowtime and utilization. Flowtime metric is important to measure the response time to the user submissions of task executions, while utilization metric measures the resource utilization. Maximizing the resource utilization of the computational grid system is gaining importance due to the economic aspects of computational grid systems (Xhafa & Abraham, 2010).

3.3 Summary

To achieve the main objective, a framework has been proposed in this study. Experimental research method has been adopted in conducting this research. GA and TS have been employed to solving the stagnation problem in ACS. A simulator has been proposed to simulate the job scheduling algorithm in the dynamic and static grid computing system. A comprehensive evaluation has been proposed to evaluate the performance of the proposed hybrid ACS algorithm using the standard performance metrics.

CHAPTER FOUR

SIMULATOR DEVELOPMENT

This chapter presents the development step for grid computing environment simulator. All the proposed algorithms plus the algorithms used for evaluations are implemented in this simulator.

This chapter is organized as follows: Section 4.1 presents the measurement criteria. The implementation of benchmark problems is discussed in Section 4.2. Simulator verification and validation is provided in Section 4.3. Finally, the chapter is summarized in Section 4.4.

4.1 Identifying the Measurement Criteria

The problem in job scheduling for grid computing is known as multi-objective problem due to the various criteria in computational grid, such as makespan, flowtime, load balancing, utilization, matching proximity, turnaround time, total weighted completion time, and average weighted response time (Xhafa & Abraham, 2008a). In this study, three metrics were implemented with priority to makespan as the main optimization objective. Makespan metric measures the general productivity of the grid computing. The best scheduling algorithm is the one that can produce a small value of makespan, which means that the algorithm is able to map tasks to machines in a good and efficient way. Therefore, the main objective in this study is to minimize the makespan.

4.2 Implementing the Benchmark Problems Model

One of the main factors that affect the performance of a grid computing is the workload to which the system is subjected (Feitelson, 2013). Evaluating the job scheduling algorithm with wrong workloads will lead to erroneous results which cannot be relied upon (Smith, 2007). One of the successful models for heterogeneous static computing system is expected time to compute proposed in Braun et al. (1999). According to Xhafa (2007), ETC model, proposed by Braun et al. (2001), is the most known model to be the most difficult benchmark for static instances of the problem. Table 4.1 shows the studies which have implemented ETC model.

Table 4.1

Algorithms evaluated with ETC model.

Authors	Algorithms evaluated using ETC model
(Maheswaran et al., 1999)	MCT, MET, (switching algorithm, k-percent best, Min-min, Sufferage.
(Braun et al., 2001)	OLB, MET, MCT, Min-Min, Max-Min, Duplex, GA, SA, GSA, TS, and A^*.
(Ritchie & Levine, 2003)	Local Search.
(Ritchie & Levine, 2004)	Hybrid ACO.
(Xhafa, 2006)	Genetic algorithm.
(J.-K. Kim et al., 2007)	Max–Min, Min–Min, Max–Max, Relative Cost, Slack Sufferage, Switching Algorithm, Genetic algorithm,
(Xhafa, Duran, et al., 2008)	Genetic algorithm.
(Xhafa, Carretero, et al., 2009)	Tabu Search.
(Izakian et al., 2010)	Discrete Particle Swarm Optimization
(Kromer et al., 2009)	Differential Evolution.
(Kołodziej et al., 2011)	Enhanced Genetic algorithm.
(Kromer, Platos, & Snasel, 2012)	Artificial Immune Systems, Differential Evolution, and Genetic Algorithms.
(Rajni & Chana, 2013)	Bacterial Foraging algorithm.
(S.-S. Kim et al., 2013)	Artificial Bee Colony.

The model arranges the information in a two dimension matrix called ETC matrix. Each entry in the matrix $ETC[i,j]$ represents the expected execution time of task$[i]$ on

machine[j]. In ETC matrix, the elements along a row represent the estimates of the expected execution times of a given task on different machines, while the elements along a column give the estimates of the expected times of different tasks on a given machine.

In order to compare the implemented algorithms with other algorithms, a grid computing simulator is developed. The simulator has the ability to generate benchmark problems for static and dynamic environments. In both environments, the size of the tasks is presented in millions of instructions format. The resources capacity is presented in terms of millions of instructions per second.

For static environment, the benchmark problems are generated using ETC model proposed by Braun et al. (2001). The calculation of the matrix will be based on expected time to compute of each task with every processor element in the grid resources. The ETC matrix values are generated using range-based technique (Braun et al., 2001). The ETC matrix will be categorized into four categories as follows:

a) High task heterogeneity and high machine heterogeneity.
b) High task heterogeneity and low machine heterogeneity.
c) Low task heterogeneity and high machine heterogeneity.
d) Low task heterogeneity and Low machine heterogeneity.

Each category will be classified further into three classes: consistent, inconsistent, and semi-consistent ETC matrices. These classes are orthogonal to the previous categories. This combination produced twelve ETC matrices.

The benchmark problems based on ETC model provide a good static environment to conduct the experiments. However, in ETC matrix, the computing capacity of resources remains unchanged (static) during tasks execution. In addition, the number of available resources, resource load, and task size are fixed. Thus, ETC model does not reflect the real dynamic environment in grid computing (Xhafa & Abraham, 2008a).

According to Smith (2007), the differences between static and dynamic workloads may have subtle implications for performance evaluation. Therefore, the experiment that utilizes static workloads is incapable to evaluate the performance of job scheduling algorithm. Feitelson (2013) has also stated that static workload cannot be considered as valid samples of real dynamic workloads. Organizations such as System Performance Evaluation Consortium, Grid Workloads Archive and Transaction Processing Performance Council provide several benchmarks on CPU, network file system, web servers, cluster, grid, database and parallel distributed systems for evaluation of computer systems (Feitelson, 2013). These benchmarks are useful to be analysed and modelled.

For dynamic environment, the developed simulator has the ability to generate benchmark problems with dynamic attributes based on benchmark modelling. Benchmark modelling is the attempt to create a simple and general model which can be used to generate synthetic workload. A good benchmark model is the one which has the ability to capture the statistical pattern of the real workloads. In addition, any benchmark model should contain a tunable parameter that allows for the generation of different load conditions. Such a model should reflect the real environment which is known as trace-driven simulations (Feitelson, 2013). Therefore, the most important

factor in any model is the underlying patterns, typically in the form of probability distributions. Figure 4.1 illustrates how to model a workload from a real system (Feitelson, 2013). Users submit their tasks to the grid computing and the grid will identify the task requirements and assign them to suitable resources. After successfully processing the task, the output will be sent back to the user based on the required output type. Every transaction inside the grid will be registered in the log file (Feitelson, 2013; Gainaru, Cappello, Trausan-matu, & Kramer, 2011). However, the degree of log details is different in each grid based on its implementation (Krakov & Feitelson, 2013). From the log file, a trace data will be created for analysis and by applying statistical methods on these trace data, the histogram and distribution pattern will be discovered (Javadi, Kondo, Vincent, & Anderson, 2009; Sonmez, Yigitbasi, Abrishami, Iosup, & Epema, 2010). These patterns will be used to model the workload which reflects the real environment plus the flexible characteristics, such as the workload size and format required by the user (Feitelson, 2013; Li, Groep, Wolters, & Templon, 2006).

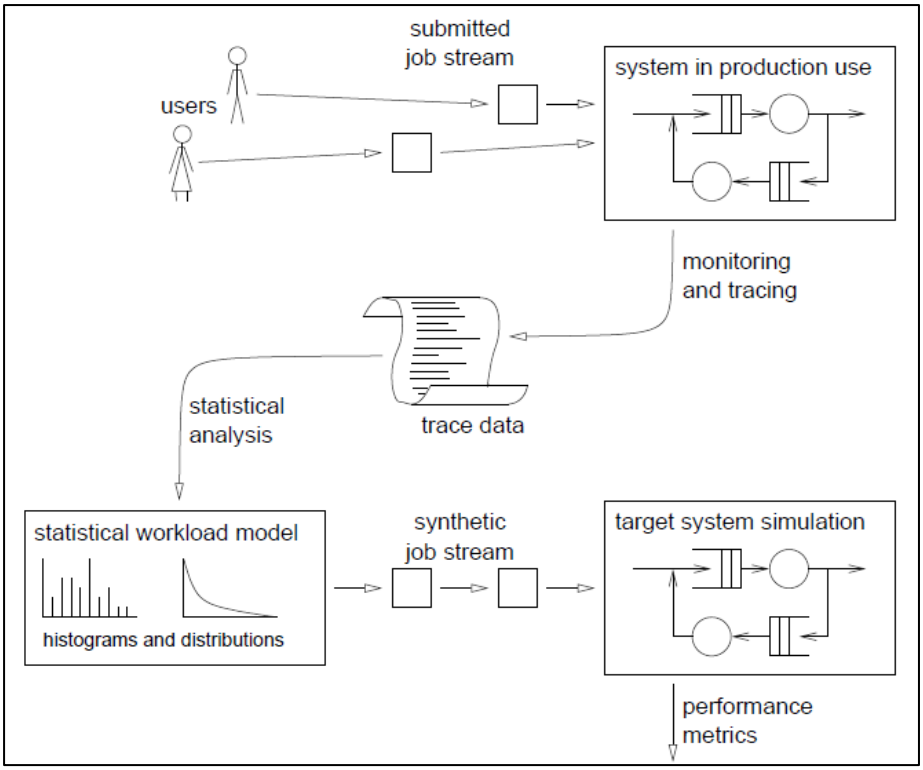

Figure 4.1. Workload modelling (Feitelson, 2013)

ETC model assumes that all machines are stable and available at all times, which is far from the real environment where machine failures are very common and inevitable (Javadi, Kondo, Iosup, & Epema, 2013). In addition, the option to add current machine load, specifying task size and machine speed are not provided in the ETC model. These features or attributes are crucial in job scheduling experiments using heuristic and metaheuristics approaches (Xhafa & Abraham, 2008a).

This study enhanced the ETC model with dynamic attributes. The enhancement to the ETC model is named Dynamic Expected Time to Compute (DETC). Statistical analyses on the jobs and resources have been performed. The proposed model uses three probability distributions, namely Normal, Gamma and Weibull to describe the nature of the resources and the jobs submitted to the grid (Carvalho & Brasileiro,

2012; A. A. Iosup, Epema, Maassen, & Nieuwpoort, 2007). The join and drop of resources implies the availability of the network connection.

Probability distributions were also used to describe the characteristics of resources in the grid (Heien et al., 2011). This will enable users with the ability to conduct a test on different types of arrival patterns for the jobs and availability patterns of the resources in validating the robustness of the scheduling algorithm. DETC model consists of three vectors and three matrices. The vectors represent the present machine load, task size and machine speed. The load is measured in second and this reflects the total time required by a machine to be ready to process the next task. The task size is measured in Millions of Instructions (MI) and the machine speed is measured in Millions of Instructions Per Second (MIPS) (Xhafa & Abraham, 2010). In static environment, $ETC[i,j]$ could be calculated simply by dividing the workload of task i by the computing capacity of resource j (Xhafa & Abraham, 2008a). However, for dynamic environment, it is proposed that the entry $[i,j]$ in DETC is calculated as follows:

$$DETC[i,j] = Load[j] + \frac{task[i]}{machine[j]} \tag{4.1}$$

The model also provides machine failure probability which follows the Weibull distribution to mimic the real environment (Iosup, Jan, Sonmez, & Epema, 2007). A sequence of benchmark with time (t) specified by the user is provided. For example, if the user specifies t =10, then the model will generate ten datasets with different machine load, different DETC and dynamic machine status. This indicates that the number of available machines is dynamic. In order to generate dynamic ETC benchmarks, the users are required to enter inputs for the parameters. These

parameters play a very important role to shape the distribution pattern that mimic the real-world grid computing environment. The proposed model generates the following attributes:

i. Load: the current load that each machine has to process before starting to process a new task. Normal distribution and Gamma distribution could be used to generate the load vector. The Gamma distribution is suitable to be used to model workload parameters (Feitelson, 2013).

ii. Task: correct distribution should be used to present task heterogeneity in the grid computing. Normal or Gamma distributions could generate various types of heterogeneity (Kolodziej, 2012).

iii. CPU speed: this is the capacity of each processor in the grid environment. Normal or Gamma distributions could be used to generate CPU capacity (Kolodziej, 2012).

iv. CPU Failure: for failure distribution, Weibull distribution is found to best represent the real-world failure in grid computing (Klusacek & Rudova, 2010).

DETC has adopted ETC matrix and integrates it with the new dynamic attributes such as machine load and machine failure. The DETC model will be very practical in testing different scheduling algorithms in an environment similar to the real grid computing system. A simulator that incorporates the DETC model has been developed using C# language in Microsoft visual studio for desktop. Figure 4.2 depicts the simulator interface which includes load, task and CPU failure settings. Appendix D provides the C# code for DETC simulator.

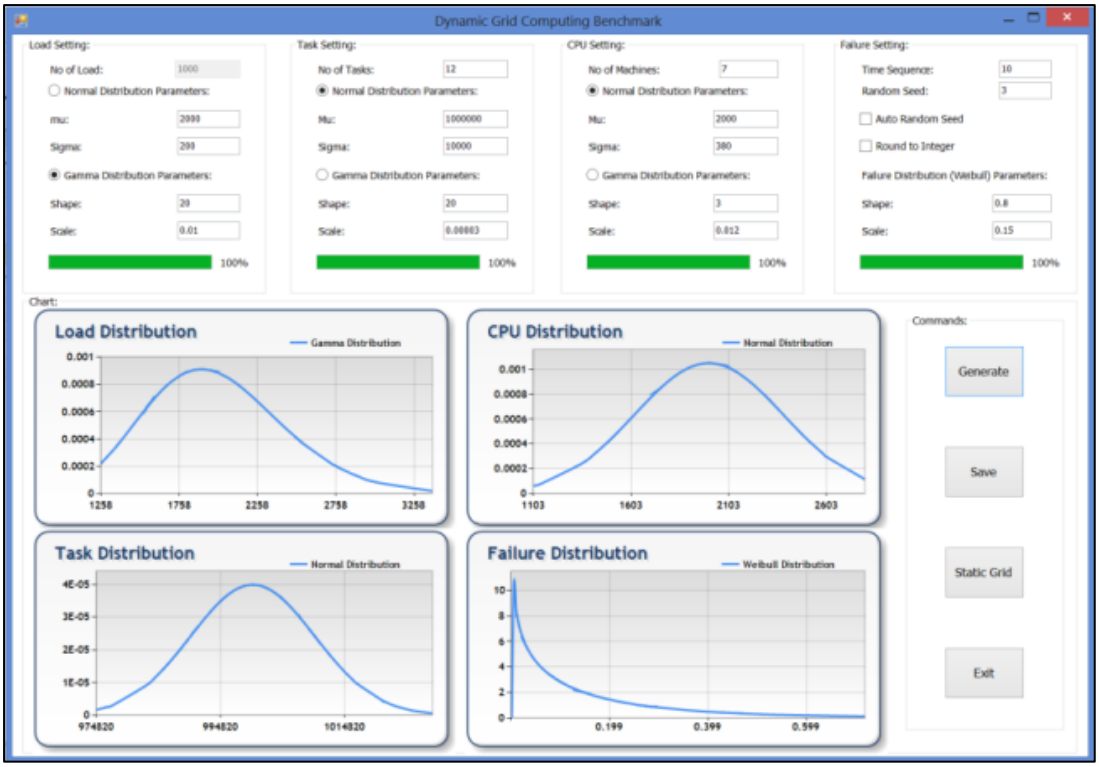

Figure 4.2. DETC simulator interface

The benchmark for tasks and resources could be created using auto seed for random benchmark or fixed seed to generate repeatable benchmark. Table 4.2 represents the parameters used to generate the benchmark in Figure 4.2.

Table 4.2

Experimental parameters

Parameter	Value
Time sequence	10
Random seed	3
Load distribution	Gamma
Load shape	20
Load scale	0.01
Task distribution	Normal
Task mean	1000000
Task standard deviation	10000
CPU distribution	Normal
CPU mean	2000
CPU standard deviation	380
CPU failure distribution	Weibull
Failure shape	0.8

Failure scale	0.15

The benchmark on tasks and resources generated using DETC could be saved in CSV file. Figure 4.3 shows parts of the generated benchmark with real values. An option is also provided to generate a benchmark with integer values.

	A	B	C	D	E	F	G	H
1	Time Sequence	10						
2	Random Seed	3						
3	Load Distribution	Gamma Distribution						
4	Load Shape	20						
5	Load Scale	0.01						
6	Task Distribution	Normal Distribution						
7	Task Mu	1000000						
8	Task Sigma	10000						
9	CPU Distribution	Normal Distribution						
10	CPU Mu	2000						
11	CPU Sigma	380						
12	CPU Failuer Distribu	Weibull Distribution						
13	Failure Shape	0.8						
14	Failure Scale	0.15						
15								
16	Sequence Number: 1							
17	Load	1973	1767	1777	1564	2128	2745	1433
18	Tasks	989193	1003604	1002442	984719	1000094	1016129	986141
19	Machines	1589	2137	2093	1419	2004	2613	1473
20	Machine Failure	0.193	0.042	0.013	0.274	0.076	0.294	0.225
21	----------							
22	DETC							
23	Inconsistent							
24		M1	M2	M3	M4	M5	M6	M7
25	T1	2595.525488	2229.889	2249.62	2261.106	2621.609	3123.566	2104.55
26	T2	2604.594714	2236.632	2256.505	2271.261	2628.8	3129.081	2114.333
27	T3	2603.863436	2236.088	2255.95	2270.443	2628.221	3128.636	2113.544
28	T4	2592.70988	2227.795	2247.482	2257.953	2619.377	3121.854	2101.513
29	T5	2602.385777	2234.99	2254.828	2268.788	2627.049	3127.738	2111.95
30	T6	2612.47703	2242.493	2262.489	2280.088	2635.05	3133.874	2122.836
31	T7	2593.604783	2228.46	2248.161	2258.955	2620.086	3122.398	2102.478
32	T8	2602.312775	2234.935	2254.773	2268.706	2626.991	3127.693	2111.872
33	T9	2603.460038	2235.788	2255.644	2269.991	2627.901	3128.391	2113.109
34	T10	2607.628697	2238.888	2258.808	2274.659	2631.206	3130.926	2117.606
35	T11	2587.492763	2223.916	2243.521	2252.111	2615.24	3118.681	2095.885
36	T12	2597.228446	2231.155	2250.913	2263.013	2622.96	3124.602	2106.387

Figure 4.3. Benchmark for dynamic grid computing

The file contains information of the values for all the parameters that have been used in generating the benchmark for jobs and resources. These include:

i. Load vector that represents the load value of each resource.

ii. Tasks vector that represents the size of each task.

iii. Machines vector that represents the capacity of each resource.

iv. Machines failure vector which represents the probability of failure for each resource.

v. DETC matrix which represents the expected time to compute with current load of the resources.

In addition, the simulator has the facilities to: generate reports, visualize the scheduling solution, and plot charts as depicted in Figures 4.4 and 4.5.

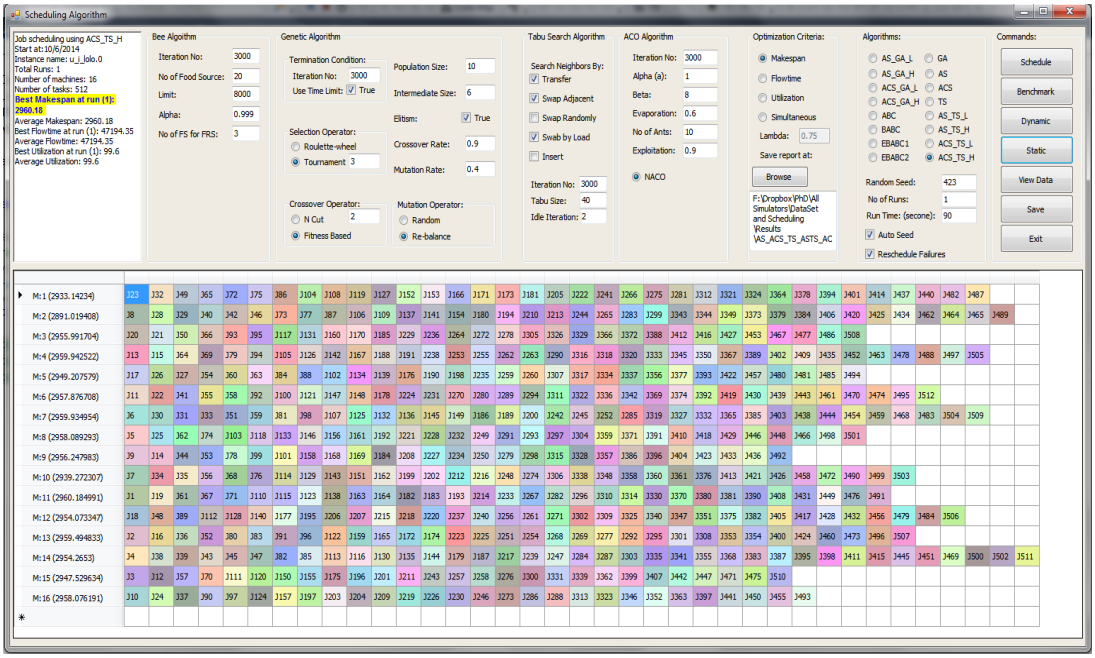

Figure 4.4. Grid computing simulator interface

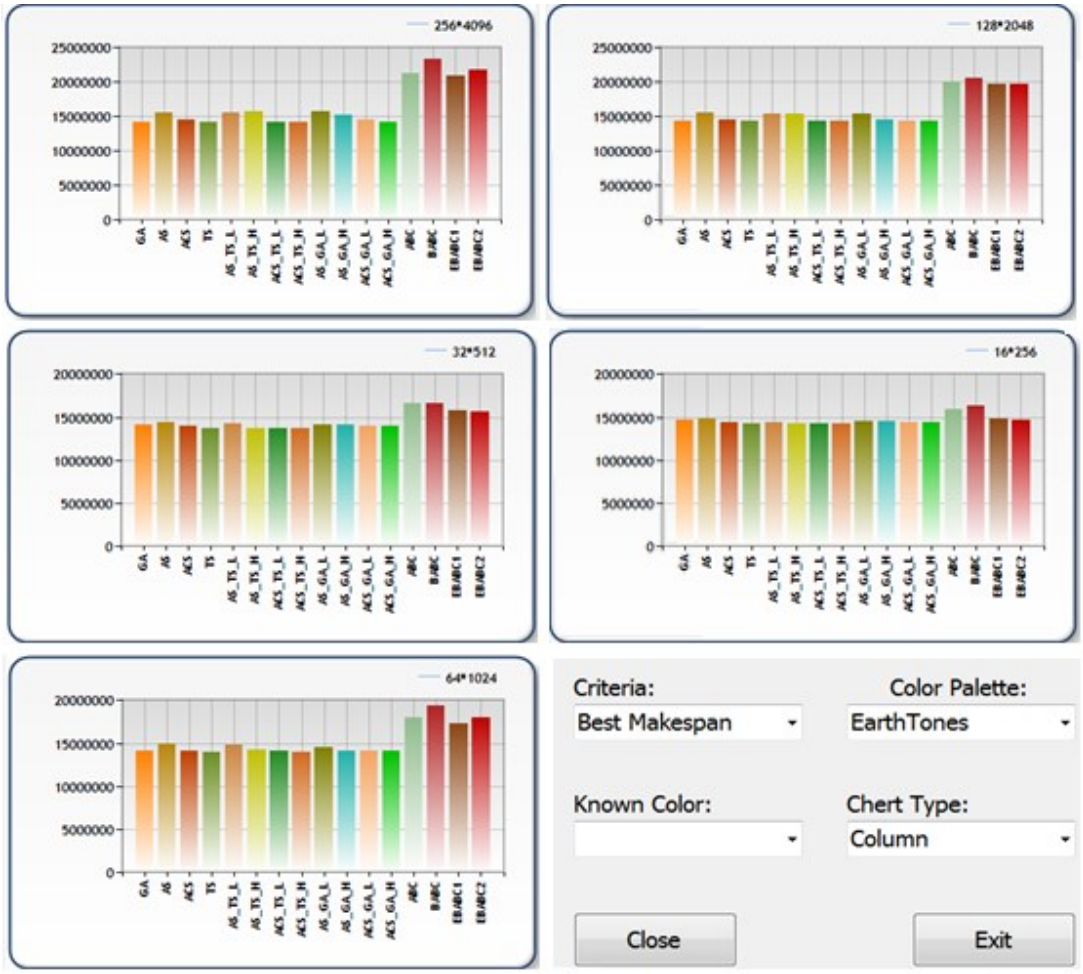

Figure 4.5. Simulator charts

4.3 Simulation Verification and Validation

A successful simulator is defined to be the one which has the ability to produce a credible and acceptable solution for the prescribed problem. However, according to Farina, Graziano, Panzieri, Pascucci, and Setola (2013), one of the biggest challenge in developing a simulator is to answer the question, "Are the results provided by the simulator believable, and, if yes, with which degree of credibility?" The authors also stated that there are several principles and techniques which have been proposed to assess the accuracy of modelling and simulation, known under the label of Verification and Validation (V&V).

Verification is defined as the process of ensuring that the simulation is implemented correctly (Garrido, 2001; Herd, Miles, McBurney, & Luck, 2014). Verification activities are generally performed concurrently with software development (Farina et al., 2013).

On the other hand, validation is defined as the process of ensuring that the simulation closely represents the real system (Garrido, 2001; Herd et al., 2014).

Verification and validation start with the initial step in this study. According to Brade and Lehmann (2002), "V&V should be associated to all phases of model development and model use". Moreover, a framework which connects all research elements substantial for V&V will help to get an overview over V&V requirements.

4.3.1 Verification Techniques

There are many tools and techniques for simulation verification. In this study, the following techniques are used to verify the simulator (Garrido, 2001):

a. Using trace: Each implemented algorithm is traced line by line with documentation.

b. Graphical Outputs: All the simulator outputs such as benchmark problems, schedule tables, and statistics are represented graphically in the simulator interface.

c. Testing with similar and different seed: Each algorithm using random variables is tested with fixed seed and random seed in order to observe the algorithm behaviours.

d. Consistency tests: The simulator is tested to generate similar results for the same parameter values.

4.3.2 Validation Techniques

Validation is another important criterion to measure the simulator quality. Validation is the answer regarding whether the assumptions taken about the real system are correct (Garrido, 2001; Herd et al., 2014). In other words, validation focuses on how much close is the behaviour of the developed simulator to the behaviour of the real system (in this study, it is the grid computing). The following techniques are adopted to validate the grid computing environment simulator (Garrido, 2001):

a. Understanding the workflow of the grid computing from literature.

b. Implementing the distribution pattern of the real grid computing log file.

c. Matching the simulation results with the required output.

4.3.3 Testing the Proposed Hybrid Algorithms

Another technique known as dynamic methods has been proposed by David (2013) for simulation verification. The proposed methods are based on exercising the implemented model with a predefined problem scenario. The solutions obtained are compared to determine whether the computerized model has been implemented appropriately. A dynamic method is applied in this study as well. In order to test the simulator calculation accuracy, a scenario consists of three resources and thirteen jobs are designed with known expected time to compute matrix (Kim et al., 2013). Table 4.3 shows the expected time to compute the jobs on different machines. Each row represents the expected time to compute job on different machines. The first column

represents the job numbers while other columns represent the expected time to compute different jobs on a machine.

Table 4.3

ETC matrix for 3 resources and 13 jobs

Jobs / Machines	M_1	M_2	M_3
J_1	1.5	2	3
J_2	3	4	6
J_3	4	5.33	8
J_4	5	6.66	10
J_5	6	8	12
J_6	7	9.33	14
J_7	7.5	10	15
J_8	9	12	18
J_9	10	13.33	20
J_{10}	10.5	14	21
J_{11}	12	16	24
J_{12}	13	18	27
J_{13}	15	20	30

The exercising experiments are conducted on all implemented algorithms. Each algorithm has executed one time with 10 seconds and random seed value is set to 58. The output of each algorithm is in the form of a schedule table which represents the final scheduling map. Each schedule table is calculated manually to determine the simulator calculation accuracy. Figure 4.6 shows the scheduling solution of the proposed ACS(TS) algorithm using the scenario presented in Table 4.3. Figure 4.6 shows three machines with different number of jobs and the grid makespan is 46.

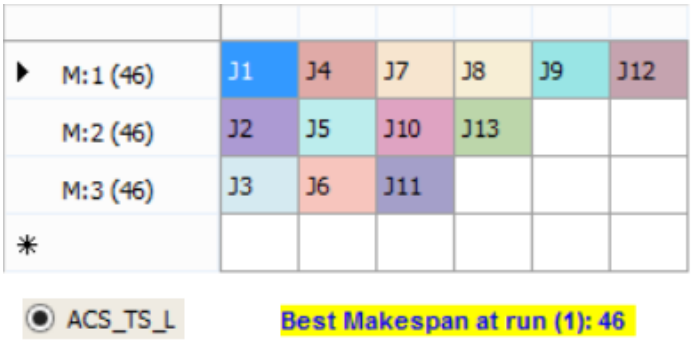

Figure 4.6. ACS(TS) Schedule table

Figure 4.7 shows the scheduling solution of the proposed ACS+TS algorithm using the scenario presented in Table 4.3. Figure 4.7 shows three machines with different number of jobs and the grid makespan is 46.

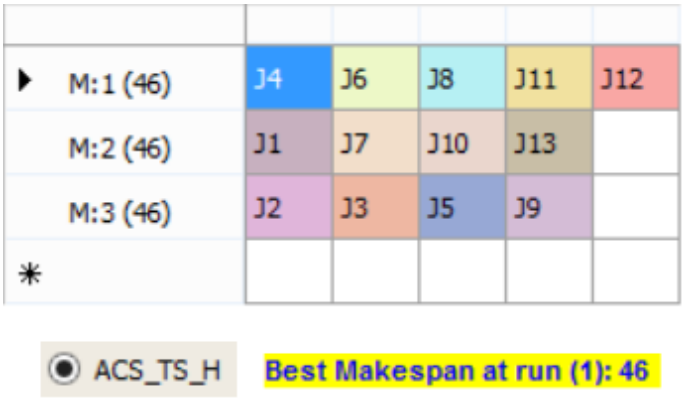

Figure 4.7. ACS+TS Schedule table

Figure 4.8 shows the scheduling solution of the proposed ACS(GA) algorithm using the scenario presented in Table 4.3. Figure 4.8 shows three machines with different number of jobs and the grid makespan is 46.67.

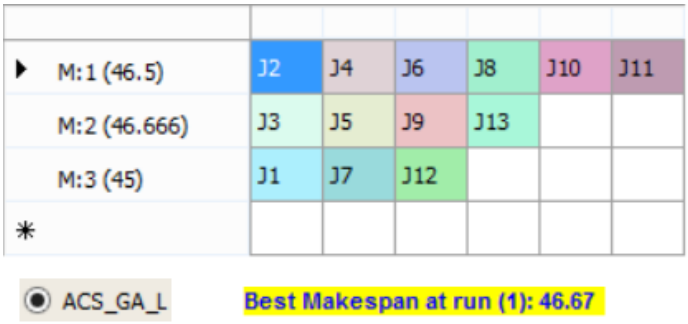

Figure 4.8. ACS(GA) Schedule table

Figure 4.9 shows the scheduling solution of the proposed ACS+GA algorithm using the scenario presented in Table 4.3. Figure 4.9 shows three machines with different number of jobs and the grid makespan is 46.67.

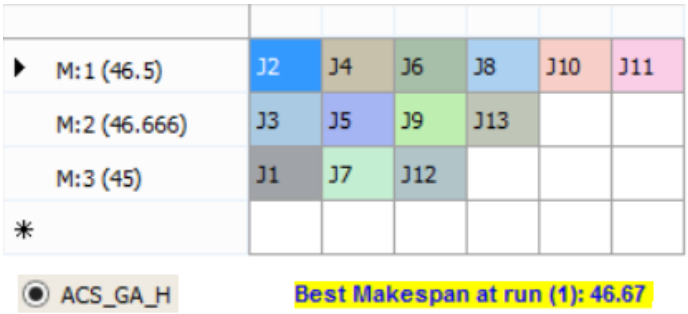

Figure 4.9. ACS+GA Schedule table

4.4 Summary

According to Pace (2003), it is impossible to check for every possible fault and bugs in a sizeable simulation; simply because there is not enough time (not even theoretically). Therefore, it is important to use a good software engineering process consistently throughout the simulator development. These processes employ simulation development environments that facilitate formal methods, and conduct thorough V&V throughout the simulation life cycle starting from the beginning. Therefore, this chapter presented the simulator development steps as well as verification and validation techniques.

CHAPTER FIVE

JOB SCHEDULING IN STATIC GRID COMPUTING

This chapter presents the evaluation and the proposed algorithm in the static grid computing environment. Section 5.1 discusses the static environment and ETC model for benchmark problems. The parameters of the proposed algorithms with the algorithms used for comparison are presented in Section 5.2. The experiment results of the proposed hybrid algorithms are provided in Section 5.3. Finally, Section 5.4 summarizes the chapter.

5.1 Static Environment

For static environment, the benchmark problems are generated using ETC model proposed by Braun et al. (2001). The calculation of the matrix will be based on expected time to compute of each task with every processor element in the grid resources. The ETC matrix has been categorized into four categories as follows:

- High task heterogeneity and high machine heterogeneity.
- High task heterogeneity and low machine heterogeneity.
- Low task heterogeneity and high machine heterogeneity.
- Low task heterogeneity and Low machine heterogeneity.

Each category has been classified further into three classes: consistent, inconsistent, and semi-consistent ETC matrices. These classes are orthogonal to the previous categories. This combination has produced twelve ETC matrices.

5.2 Algorithms Parameters

Tuning metaheuristics parameters to find the best possible configuration of the algorithm is a very critical task (Aleti, 2012). In fact, the parameters in metaheuristics algorithms have the characteristics of a machine learning problem (Birattari, 2009). In this study, the parameter values for the algorithms which are implemented for comparison are adopted from Dorigo and Stutzle (2004); Kim et al. (2013); Xhafa, Carretero, et al. (2009); and Xhafa, Duran, et al. (2008) in order to perform a fair comparison. Table 5.1 summarizes the adopted algorithms and their parameter resources.

Table 5.1

Algorithms resource for parameter values

Algorithm name	Resource
GA	(Xhafa, Duran, et al., 2008)
AS, ACS	(Dorigo & Stutzle, 2004)
TS	(Xhafa, Carretero, et al., 2009)
BABC, EBABC1, EBABS2	(Kim et al., 2013)
PSO-GELS	(Pooranian et al., 2013)

5.2.1 Genetic Algorithm Parameters

In this study, the algorithm implementation and parameter values are adopted from Xhafa, Duran, et al. (2008). Table 5.2 reports the parameter values.

Table 5.2

GA parameter values

Parameter	Value
Iteration No	3000
Time limit	90 seconds
Population size	10
Intermediate size	6
Crossover rate	0.9
Mutation rate	0.4
Selection operator	Tournament 3

Crossover operator	Fitness based
Mutation operator	Re-balance

In Table 5.2, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. The population size parameter represents the total number of solutions generated as a population for crossover and mutation operators. Part of the population is selected for crossover and mutation which is determined by the intermediate size parameter. The probability that crossover and mutation operators will be applied to the selected solution is controlled by crossover and mutation rate parameters respectively. Selection operator is very important to keep the diversity between solutions in genetic algorithm. The solutions selected using an operator is known as tournament operator which uses three candidates. This selection mechanism will ensure that each solution has a chance to compete with the other two solutions fairly. After the selection process, each pair of solutions applies crossover using fitness based operator. Finally, the mutation operator is applied using re-balanced operator in order to keep the scheduling balanced.

5.2.2 Ant System Parameters

The parameter values for AS algorithm in this study are adopted from Dorigo and Stutzle (2004). Table 5.3 provides the parameter values for AS algorithm.

Table 5.3

AS parameter value

Parameter	Value
Iteration No	3000
Alpha	1
Beta	8

Evaporation rate	0.5
No of ants	No of machines
Time limit	90 seconds

In Table 5.3, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. The alpha parameter represents the pheromone influence, increasing the alpha value which will influence the ants to rely more on pheromone instead of heuristic value and vice versa. Beta parameter has the influence on heuristic value. Increasing the beta value will force the ants to follow the heuristic values strongly, while reducing its value will influence the ants more towards pheromone value. The duration of pheromone value is determined by the evaporation rate parameter. If the evaporation value is one, means all the pheromones will be evaporated after each cycle. In opposite, if the evaporation value is zero, then no evaporation will occur at all. In AS algorithm, each ant constructs its own solution, whereby the total number of ants is specified by the number of ant parameter.

5.2.3 Ant Colony System Parameters

ACS algorithm parameters values are adopted from Dorigo and Stutzle (2004). Table 5.4 shows the parameter values.

Table 5.4

ACS parameter values

Parameter	**Value**
Iteration No	3000
Beta	8
Evaporation rate	0.6
No of ants	10
Exploitation	0.9
Time limit	90 seconds

In Table 5.4, the parameters iteration number, beta, evaporation rate, number of ants, and time limit are defined the same as in ant system algorithm. The extra parameter in ACS algorithm is the exploitation rate parameter. This parameter controls the algorithm exploitation/exploration behaviour. The value one means that the algorithm will do exploitation search in a greedy way, while the value 0 means the ants will search the space randomly.

5.2.4 Tabu Search Parameters

This study adopts the values for tabu search algorithm parameters from Xhafa, Carretero, et al. (2009). Table 5.5 shows the parameter values for TS algorithm.

Table 5.5

TS parameter values

Parameter	Value
Iteration No	3000
Search neighbour by	Transfer, Swap adjacent, and Swap by load
Tabu size	40
Idle iteration	2
Time limit	90 seconds

In Table 5.5, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. One of the most important parameters in tabu search is searching the neighbourhood of the current solution. There are many techniques to perform this search. Three methods are used in this study to search the neighbourhood. First, transfer method works based on transferring between different jobs and machines. Second, swap adjacent works based on interchanging the adjacent jobs in the solution vector. Third, swap by load will interchange between high and

low load machines. The tabu size parameter will specify how many moves to keep in the list as a tabu move. TS algorithm will change from soft diversification into strong diversification based on the idle iteration parameter.

5.2.5 BABC, EBABC1, and EBABC2 Parameters

These algorithm implementation and parameter values are adopted from Kim et al. (2013). Table 5.6 reports the parameter value for each algorithm.

Table 5.6

BABC, EBABC1, EBABC2 parameter values

Parameter name	BABC	EBABC1	EBABC2
Iteration No	3000	3000	3000
Number of food source	20	20	20
Limit	800	800	800
Time limit (seconds)	90	90	90
No of FRS	NA	3	NA
Alpha	NA	NA	0.999

NA: Not applicable for specific algorithm.

In Table 5.6, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. The number of food source parameter specifies the number of solutions which is equal to the number of employed bees. The parameter limit controls the number of trials to improve the solution. After reaching the limit, the source food is abandoned and the employed bee for that food source becomes a scout. The extra parameters are number of FRS and alpha as proposed by Kim et al. (2013). FRS value is used to incorporate a Flexible Ranking Strategy (FRS) to improve the balance between exploration and exploitation. Alpha is a real number parameter greater than 0 and less than 1.

5.2.6 PSO-GELS Parameters

The parameters values for Particle Swarm Optimization and Gravitational Emulation Local Search (GELS-PSO) are adopted from the study proposed by Pooranian et al. (2013) as shown in Table 5.7.

Table 5.7

PSO-GELS Algorithm Parameters Values

Parameter	Value
Iteration No	3000
Particle No	50
V_max	40
C1	2
C2	2
GC	6.672
Radius	10

In Table 5.7, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. One of the most important parameters in PSO-GELS is the particle number which represents the number of solutions. The velocity values are initialized with maximum value determined by V-max parameters. C1 and C2 are positive acceleration constants which control the influence of the best and neighbour solutions on the search process (Izakian et al., 2010). GC is a constant with the value 6.672 and Radius is the neighbour radius between the two responses in the search space.

5.2.7 AS(TS), AS+TS, ACS(TS), and ACS+TS Parameters

Based on the stand-alone version of these algorithms, the hybrid approaches utilized the same parameter values as shown in Table 5.8.

Table 5.8

AS, ACS, and TS Algorithms Parameter Values

Parameter name	AS	ACS	TS
Iteration No	3000	3000	3000
Alpha	1	1	NA
Beta	8	8	NA
Evaporation	0.5	0.6	NA
No of ants	No of machines	10	NA
Exploitation	NA	0.9	NA
Time limit (seconds)	45	45	45
Search neighbour	NA	NA	Transfer, Swap adjacent, and Swap by load
Tabu size	NA	NA	40
Idle iteration	NA	NA	2

NA: Not applicable for specific algorithm.

5.2.8 AS(GA), AS+GA, ACS(GA), and ACS+GA Parameters

These hybrid approaches adopted their parameter values from the stand-alone versions of them. Tables 5.9 and 5.10 summarize the parameter values.

Table 5.9

AS(GA) and AS+GA Algorithms Parameter Values

AS		**GA**	
Parameter name	**Value**	**Parameter name**	**Value**
Iteration No	3000	Iteration No	3000
Alpha	1	Time limit	45 seconds
Beta	8	Population size	10
Evaporation	0.5	Intermediate size	6
No of ants	Number of machines	Selection operator	Tournament 3
Time limit	45 seconds	Crossover rate	0.9
		Mutation rate	0.4
		Crossover operator	Fitness based
		Mutation operator	Re-balance

Table 5.10

ACS(GA) and ACS+GA Algorithms Parameter Values

ACS		GA	
Parameter name	**Value**	**Parameter name**	**Value**
Iteration No	3000	Iteration No	3000
Alpha	1	Time limit	45 seconds
Beta	8	Population size	10
Evaporation	0.7	Intermediate size	6
No of ants	10	Selection operator	Tournament 3
Exploration rate	0.9	Crossover rate	0.9
Time limit	45 seconds	Mutation rate	0.4
		Crossover operator	Fitness based
		Mutation operator	Re-balance

The execution time for GA, AS, ACS, TS, BABC, EBABC1, EBABC2 was set to 90 seconds which is commonly used as a reasonable amount of time for scheduling jobs in a computational grid environment (Xhafa, Barolli, et al., 2007a; Xhafa & Duran, 2008). To keep this restriction time in the hybrid approaches, each algorithm is limited to 45 seconds in order to conduct fair experiments. However, each algorithm is also terminated with an iteration number which also works as a termination criterion besides the limited time. In other words, each algorithm is terminated either by reaching the allowed run time or reaching the allowed iteration number.

5.3 Experimental Result and Analysis

The experiments are conducted using Intel® Core (TM) i7-3612QM CPU @ 2.10 GHz and 8G RAM. The grid computing simulator is developed using visual C#. The time given for each experiment is 90 seconds (45 seconds for each algorithm in the hybrid approach). Each algorithm is executed 10 times in order to calculate the best and average values. The proposed algorithms are evaluated based on makespan, flowtime, and utilization metrics.

5.3.1 Best Makespan Results

In order to compare and represent the performance of the proposed algorithms visually, a geometric mean is calculated to normalize the makespan, flowtime, and utilization values of the twelve instances (Izakian, Abraham, & Snsel, 2009). In addition, the difference between the ACS algorithm and the proposed hybrid algorithms are calculated to provide the enhancement of each hybrid algorithm in terms of percentage.

Figure 5.1 displays the results of the sixteen algorithms in terms of the best makespan value. The Figure 5.1 shows that the worse performance produced by BABC and PSO-GELS algorithms. The algorithms EBABC2, GA, ACS, AS, and ACS(GA) show similar performance to each other which is better than BABC and PSO-GELS algorithms. The performance enhanced slightly by TS, AS(TS), ACS+GA, AS(GA), EBABC1, AS+GA, ACS(TS), and AS+TS algorithms. The proposed hybrid algorithm ACS+TS achieved the best performance in terms of best makespan value as shown in Figure 5.1. This is due to the refinement process achieved by TS algorithm to the best solution produced by ACS algorithm.

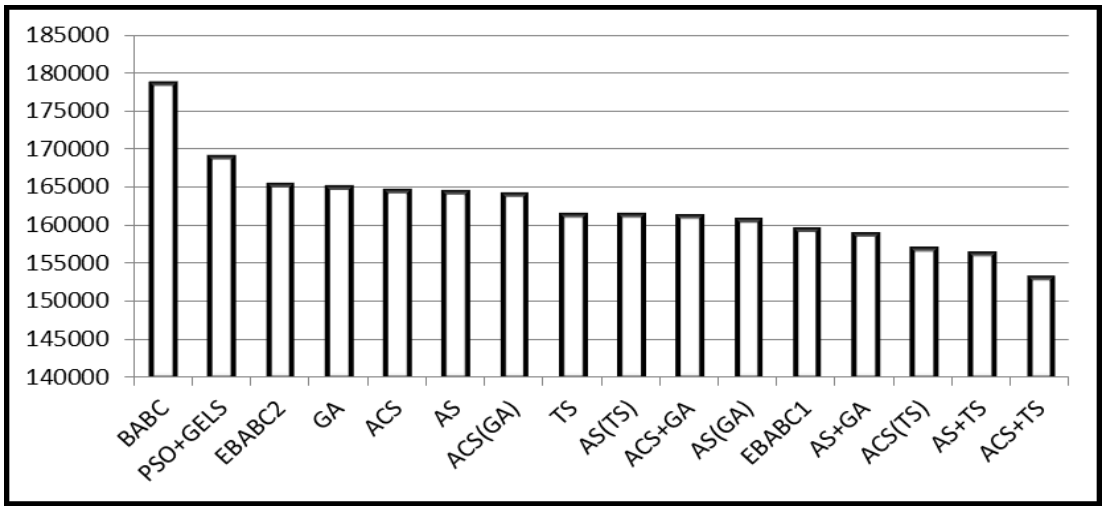

Figure 5.1. Geometric mean for the best makespan values

Figure 5.2 represents the enhancement of each hybrid algorithm which is expressed in terms of percentage. Each hybrid algorithm is compared with the ACS algorithm in terms of best makespan value enhancement. The Figure 5.2 shows that ACS+TS algorithm enhanced 6.99% followed by ACS(TS) 4.65%, ACS+GA 2.03%, and ACS(GA) 0.35%. This enhancement indicates that GA and TS algorithms increased the performance of ant colony system algorithm in both low and high levels. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in high level order. Clearly in high level order, the TS algorithm starts with initial solution passed from ACS algorithm which is very high quality produced by the best ant. Therefore, TS starts from good location in the search space which leads to further enhancement to the solution found by ACS algorithm.

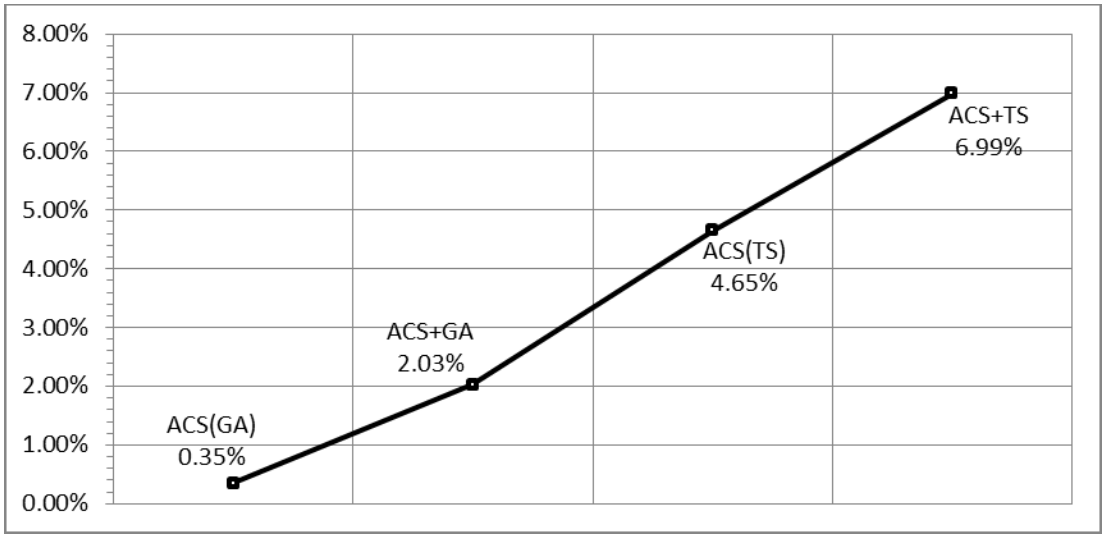

Figure 5.2. The percentage enhancement of each hybrid algorithm in terms of the best makespan values

5.3.2 Average Makespan Results

Figure 5.3 displays the results of the sixteen algorithms in terms of the average makespan value. The Figure 5.3 shows that the worse performance produced by BABC followed by EBABC2 and PSO-GELS algorithms. The algorithms ACS,

EBABC1, ACS(GA), AS, GA, ACS+GA, AS(TS), and AS(GA) show similar performance to each other which is better than BABC, EBABC2 and PSO-GELS algorithms. The performance enhanced slightly by AS+GA, TS, ACS(TS), and AS+TS algorithms. The proposed hybrid algorithm ACS+TS achieved the best performance in terms of average makespan value as shown in Figure 5.3. This is due to the refinement process achieved by TS algorithm to the best solution produced by ACS algorithm.

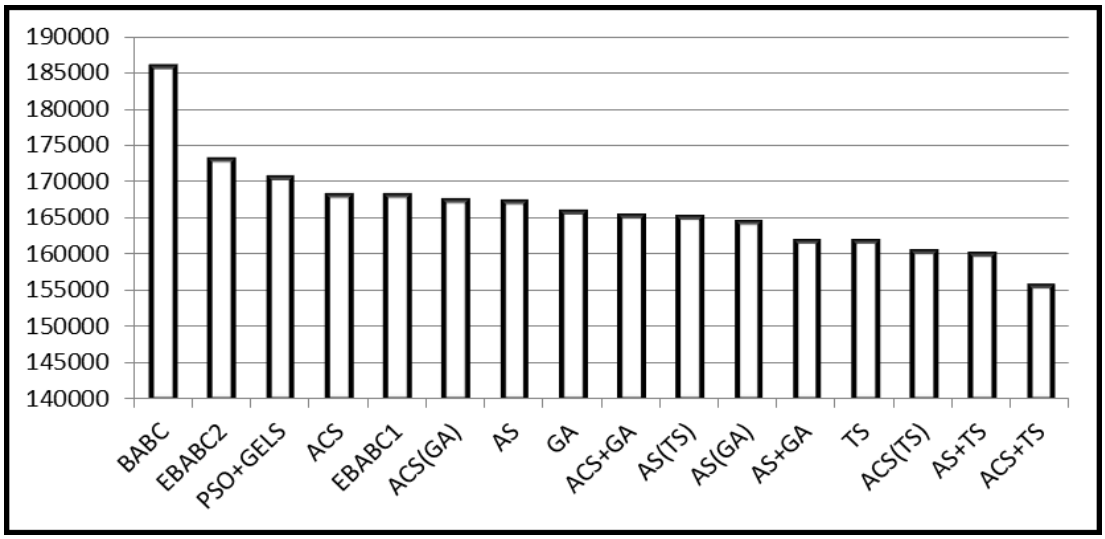

Figure 5.3. Geometric mean for the average makespan values

Figure 5.4 represents the enhancement of each hybrid algorithm which is expressed in terms of percentage. Each hybrid algorithm is compared with the ACS algorithm in terms of average makespan value enhancement. The Figure 5.4 shows that ACS+TS algorithm enhanced 7.46% followed by ACS(TS) 4.66%, ACS+GA 1.7%, and ACS(GA) 0.47%. This enhancement indicates that GA and TS algorithms increased the performance of ant colony system algorithm in both low and high levels. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in high level order. Clearly in high level order, the TS algorithm starts with initial solution passed from ACS algorithm which is very high quality produced by

the best ant. Therefore, TS starts from good location in the search space which leads to further enhancement to the solution found by ACS algorithm.

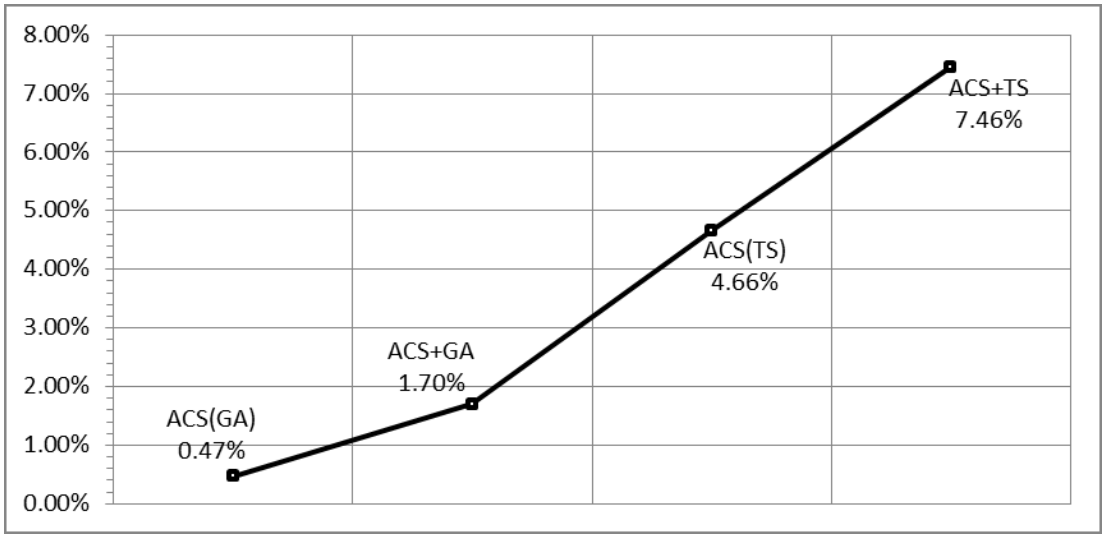

Figure 5.4. The percentage enhancement of each algorithm in terms of the average makespan values

5.3.3 Best Flowtime Results

The second metric to measure the scheduling algorithm performance is the flowtime. Figure 5.5 displays the results of the sixteen algorithms in terms of the best flowtime value. The Figure 5.5 shows that the worse performance produced by BABC followed by PSO-GELS, GA, and ACS(GA) algorithms. The algorithms ACS, TS, ACS+GA, and EBABC2 show similar performance to each other which is better than BABC and PSO-GELS algorithms. The performance enhanced slightly with AS(GA), AS(TS), AS+GA, AS, ACS(TS), AS+TS, and EBABC1 algorithms. The proposed hybrid algorithm ACS+TS achieved the best performance in terms of best flowtime value as shown in Figure 5.5. This is due to the refinement process achieved by TS algorithm to the best solution produced by ACS algorithm.

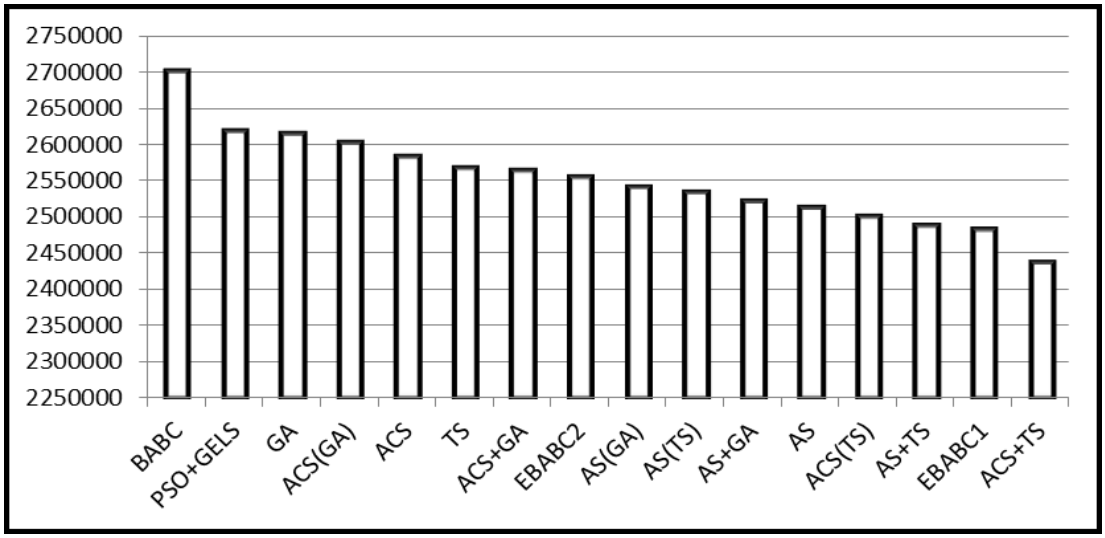

Figure 5.5. Geometric mean for best flowtime values

Figure 5.6 represents the enhancement of each hybrid algorithm which is expressed in terms of percentage. Each hybrid algorithm is compared with the ACS algorithm in terms of best flowtime value enhancement. The Figure 5.6 shows that ACS+TS algorithm enhanced 5.61% followed by ACS(TS) 3.15%, and ACS+GA 0.75%. The proposed hybrid algorithm ACS(GA) perform worse than ACS algorithm (not included in the graph). This enhancement indicates that TS algorithm increased the performance of ant colony system algorithm in both low and high levels while GA algorithm only increases the ACS algorithm performance in high level order. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in high level order.

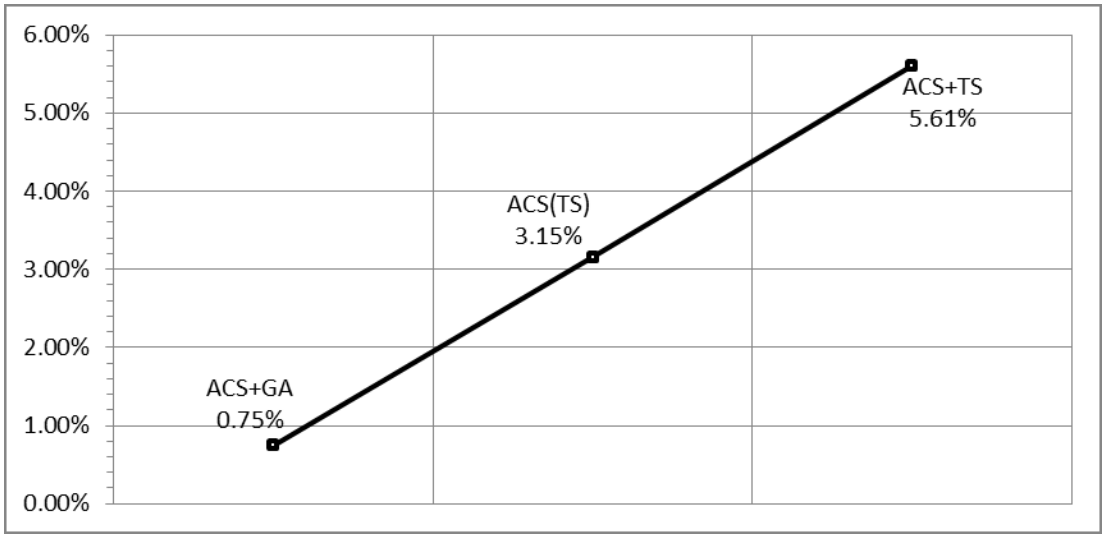

Figure 5.6. The percentage enhancement of each algorithm in terms of the best flowtime values

5.3.4 Average Flowtime Results

Figure 5.7 displays the results of the sixteen algorithms in terms of the average flowtime values. The Figure 5.7 shows that the worse performance achieved by BABC followed by EBABC2, and ACS(GA) algorithms. The algorithms ACS, ACS+GA, GA, PSO-GELS, and EBABC1 show similar performance to each other which is better than BABC algorithm. The performance enhanced little with AS(TS), TS, AS+GA, AS, ACS(TS), and AS+TS algorithms. The proposed hybrid algorithm (ACS+TS) achieved the best value in terms of average flowtime values as shown in Figure 5.7.

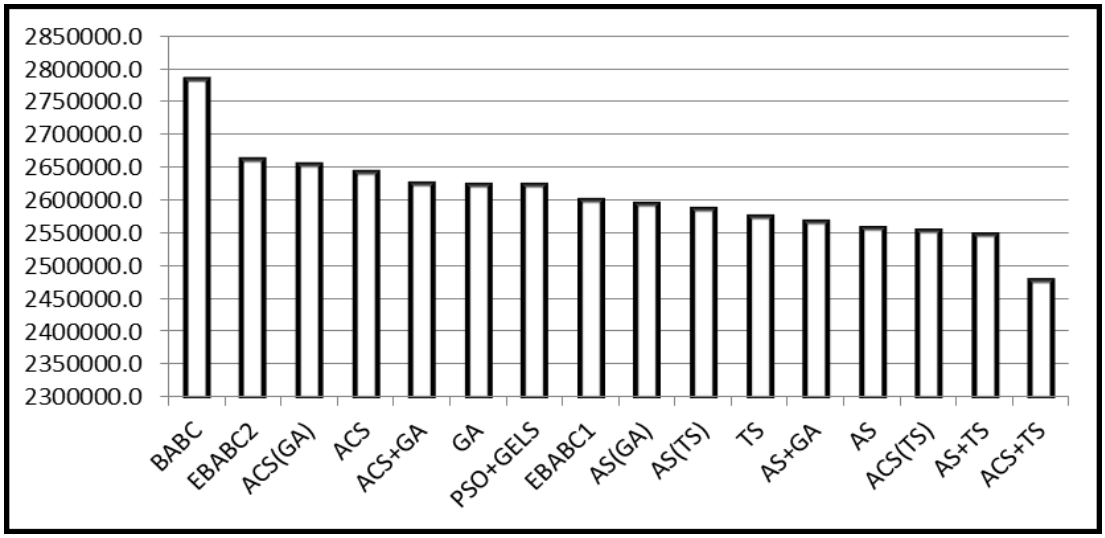

Figure 5.7. Geometric mean for average flowtime values

Figure 5.8 represents the enhancement of each hybrid algorithm which is expressed in terms of percentage. Each hybrid algorithm is compared with the ACS algorithm in terms of the average flowtime enhancement. The Figure 8.5 shows that ACS+TS algorithm enhanced 6.22% followed by ACS(TS) 3.37%, and ACS+GA 0.65%. The proposed hybrid algorithm ACS(GA) perform worse than ACS algorithm (not included in the graph). This enhancement indicates that TS algorithm increased the performance of ant colony system algorithm in both low and high levels while GA algorithm only increases the ACS algorithm performance in high level order. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in high level order.

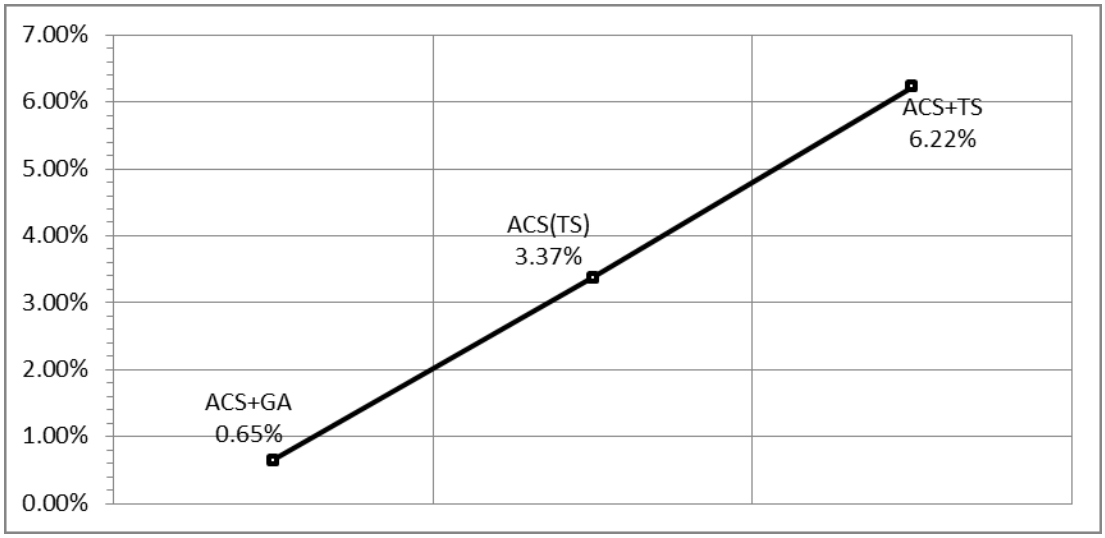

Figure 5.8. The percentage enhancement of each algorithm in terms of the average flowtime values

5.3.5 Best Utilization Results

The last metric implemented in this study to measure the algorithms performance is the utilization of resources in the computational grid.

Figure 5.9 displays the results of the sixteen algorithms in terms of the best utilization value. The Figure 5.9 shows that the worse performance achieved by BABC and AS algorithms. The algorithms PSO-GELS, EBABC2, EBABC1, AS(TS), and ACS show similar performance to each other which is better than BABC and AS algorithms. The performance enhanced little with AS(GA), GA, ACS(GA), AS+GA, ACS+GA, TS, ACS+TS, and AS+TS algorithms. The proposed hybrid algorithm ACS(TS) achieved the best value in terms of best utilization as shown in Figure 5.9. This is due to the refinement process achieved by TS algorithm to the solution produced by ACS algorithm. However, the algorithms ACS+TS, AS+TS, and ACS(TS) show similar performance in terms of best utilization value.

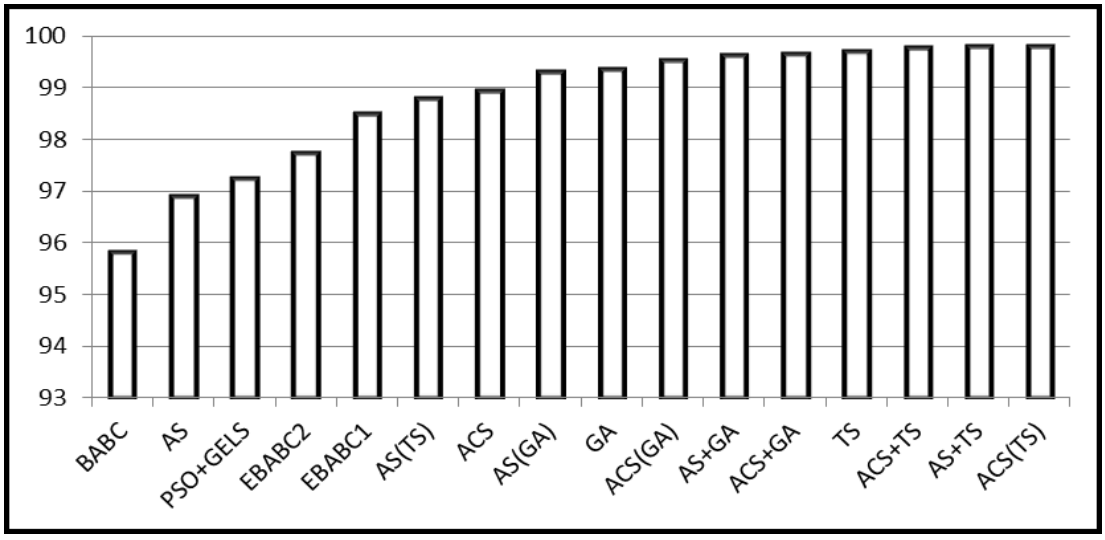

Figure 5.9. Geometric mean for best utilization value

Figure 5.10 represents the enhancement of each hybrid algorithm which is expressed in terms of percentage. Each hybrid algorithm is compared with the ACS algorithm in terms of best utilization value enhancement. The Figure 5.10 shows that ACS(TS) algorithm enhanced 0.88% and ACS+TS 0.88% followed by ACS+GA 0.47%, and ACS(GA) 0.61%. This enhancement indicates that GA and TS algorithms increased the performance of ant colony system algorithm in both low and high levels. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in both level order. Clearly in high level order, the TS algorithm starts with initial solution passed from ACS algorithm which is very high quality produced by the best ant. Therefore, TS starts from good location in the search space which leads to further enhancement to the solution found by ACS algorithm. While for low level hybridization between ACS and TS, the TS algorithm enhance the best solution at the end of each cycle which makes the ants update the pheromone based on the enhanced solution produced from TS algorithm. Therefore, the low level hybridization algorithm was able to achieve good results.

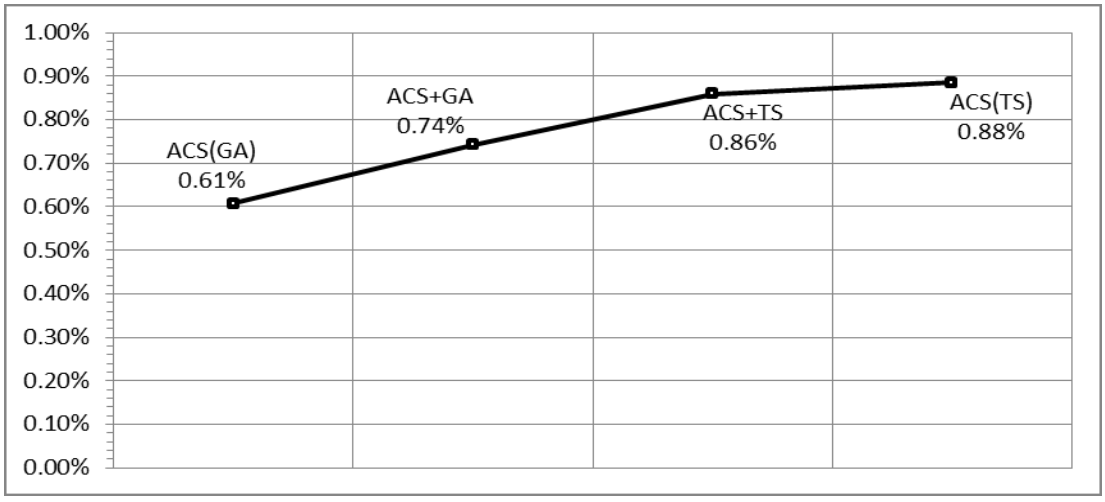

Figure 5.10. The percentage enhancement of each algorithm in terms of the best utilization values

5.3.6 Average Utilization Results

Figure 5.11 displays the results of the sixteen algorithms in terms of the average utilization value. The Figure 5.11 shows that the best performance achieved by the proposed hybrid ACS(TS) algorithm. Similar performance also achieved by ACS(TS), TS, AS(TS), ACS(GA), AS+GA, and ACS(GA) algorithms. Slightly less performance achieved by GA, AS(GA), ACS, and AS(TS) algorithms. The worse performance achieved by EBABC1, PSO-GELS, EBABC2, AS, and BABC algorithms.

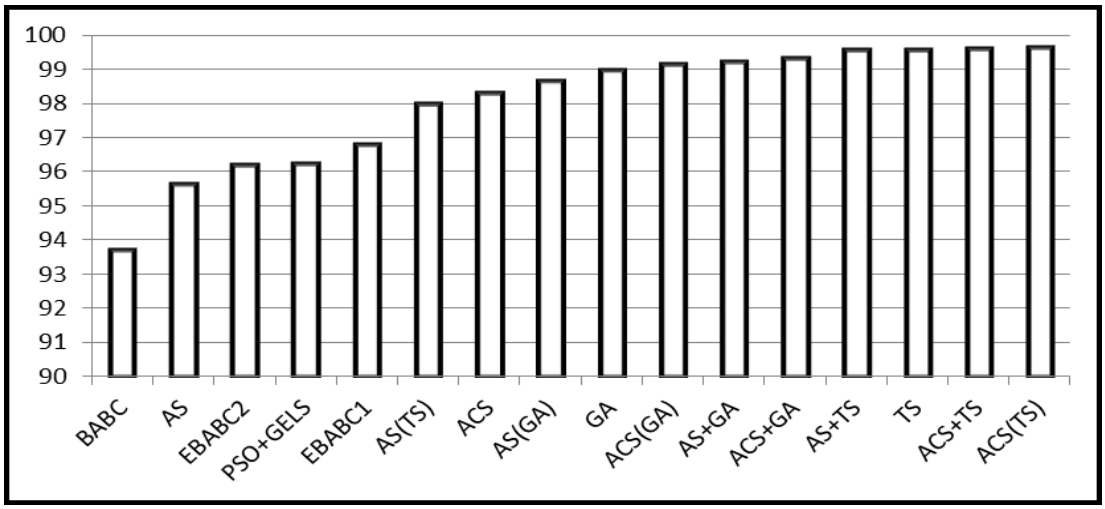

Figure 5.11. Geometric mean for average utilization values

Figure 5.12 represents the enhancement of each hybrid algorithm which is expressed in terms of percentage. Each hybrid algorithm is compared with the ACS algorithm in terms of average utilization value enhancement. The Figure 5.12 shows that ACS(TS) algorithm enhanced 1.36% followed by ACS+TS 1.33%, ACS+GA 1.07%, and ACS(GA) 0.87%. This enhancement indicates that GA and TS algorithms increased the performance of ant colony system algorithm in both low and high levels. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in low level order.

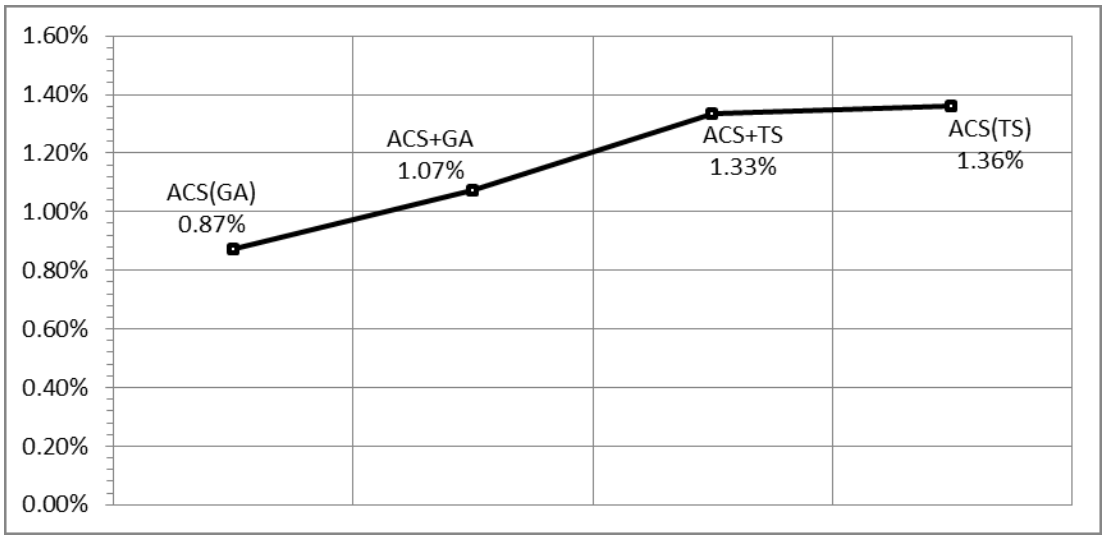

Figure 5.12. The percentage enhancement of each algorithm in terms of the average utilization values

5.3.7 Discussion

In terms of best makespan value, the proposed algorithm ACS+TS outperform the other algorithms. The best makespan value is the main objective of this study which reflects the productivity of grid computing system. ACS+TS achieved good results due to the ability of TS algorithm to refine the best solution found by ACS algorithm. The worse performances in terms of best makespan value were achieved by BABC and PSO-GELS algorithms. BABC is the binary version of ABC algorithm and PSO-

GELS is the discrete version of PSO algorithm hybridized with GELS algorithm. These type of algorithms were originally developed to solve continues problems. In spite of several studies proposed methods to apply these algorithm into discrete problems, the difficulty there is that the notions of velocity and direction have no natural extensions for combinatorial problems, such as travelling salesman problem and scheduled problems (Poli, Kennedy, & Blackwell, 2007). Moreover, hybridizing PSO with GELS did not enhance PSO algorithm, according to Gauci, Dodd, and Groß (2012) "gravitational search algorithm' is not genuinely based on the law of gravity", therefore, it could not enhance PSO algorithm.

5.4 Summary

The benchmark problems were generated using expected time to compute model consists of twelve datasets. Experimented results show that the proposed hybrid ACS+TS algorithm outperforms other algorithms in terms of makespan and flowtime criteria. However, for utilization criterion, two hybrid algorithms show good performance, namely ACS(TS) and ACS+TS.

The results confirm that hybridizing ant colony system with tabu search algorithm is very useful and efficient, specifically the high level hybridization. Tabu search algorithm was able to enhance the solution found by ant colony system algorithm. In addition, ant system also benefited from tabu search and genetic algorithm in some instances.

CHAPTER SIX

JOB SCHEDULING IN DYNAMIC GRID COMPUTING

This chapter presents the evaluation and the proposed algorithm in the dynamic grid computing environment. Section 6.1 discusses the dynamic environment and DETC model for benchmark problems. The parameters of the proposed algorithms with the algorithms used for comparison are presented in Section 6.2. The experiment results of the proposed hybrid algorithms are provided in Section 6.3. Finally, Section 6.4 summarizes the chapter.

6.1 Dynamic Environment

In spite of the effectiveness of using static environment for testing the algorithm performance, there are some attributes which are not presented in the static scenario. Attributes such as availability and varying in resource speed are presented in the dynamic scenario. In order to investigate the proposed algorithms performance, twelve algorithms are implemented for comparison. For dynamic experiment, five datasets are generated using the DETC simulator in order to mimic the real grid computing situations. Each algorithm is executed ten times on each dataset to calculate the best, average, and standard deviation values. Three metrics are implemented to measure the performance, namely makespan, flowtime, and utilization with priority to makespan as the main objective. Table 6.1 shows the dataset description (Xhafa, Koodziej, et al., 2011).

Table 6.1

Datasets descriptions

	Mini	**Small**	**Medium**	**Large**	**Very large**
Tasks	256	512	1024	2048	4096
Machines	16	32	64	128	256

In the Table 6.1, the columns represent the size of the benchmarks dataset problem. The first row represents the number of tasks in each dataset while the second row represents the number of machines in each dataset.

For benchmark problems, the simulator started by generating tasks and machines using the parameters given in Table 6.2.

Table 6.2

Parameters for Generating Dynamic Benchmark

Parameter name	**Value**
Time Sequence	1
Round to integer	True
Failure distribution	Weibull(0.8, 0.4)
No of machines	16, 32, 64, 128, 256
Machine distribution	Normal(1000, 175)
No of tasks	256, 512, 1024, 2048, 4096
Load distribution	Normal(625000, 93750)

In Table 6.2, the parameter time sequence is the number of batches to be sent to the grid computing system. The parameter round to integer makes all generated values integer if the parameter set to true. The parameter failure distribution controls the shape of the Weibull distribution method. The number of machines specifies the resources in the grid system. Machine distribution controls the shape of the Normal distribution method. The number of tasks specifies the number of tasks submitted to

the grid system. Load distribution controls the shape of the Normal distribution method.

6.2 Algorithm Parameters

Tuning metaheuristics parameters to find the best possible configuration of the algorithm is a very critical task (Aleti, 2012). In fact, the parameters in metaheuristics algorithms have the characteristics of a machine learning problem (Birattari, 2009). In this study, the parameter values for the algorithms which are implemented for comparison are adopted from Dorigo ad Stutzle (2004); Kim et al. (2013); Pooranian et al.(2013) Xhafa, Carretero, et al. (2009); and Xhafa, Duran, et al. (2008) in order to perform a fair comparison. Table 6.3 summarizes the adopted algorithms and their parameter resources.

Table 6.3

Algorithms resource for parameter values

Algorithm name	Resource
GA	(Xhafa, Duran, et al., 2008)
AS, ACS	(Dorigo & Stutzle, 2004)
TS	(Xhafa, Carretero, et al., 2009)
BABC, EBABC1, EBABS2	(Kim et al., 2013)
PSO-GELS	(Pooranian et al., 2013)

6.2.1 Genetic Algorithm Parameters

In this study, the algorithm implementation and parameter values are adopted from Xhafa, Duran, et al. (2008). Table 6.4 reports the parameter values.

Table 6.4

GA parameter values

Parameter	Value
Iteration No	3000
Time limit	90 seconds
Population size	10
Intermediate size	6
Crossover rate	0.9
Mutation rate	0.4
Selection operator	Tournament 3
Crossover operator	Fitness based
Mutation operator	Re-balance

In Table 6.4, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. The population size parameter represents the total number of solutions generated as a population for crossover and mutation operators. Part of the population is selected for crossover and mutation which is determined by the intermediate size parameter. The probability that crossover and mutation operators will be applied to the selected solution is controlled by crossover and mutation rate parameters respectively. Selection operator is very important to keep the diversity between solutions in genetic algorithm. The solutions are selected using an operator known as tournament operator using three candidates. This selection mechanism will ensure that each solution has a chance to compete with the other two solutions fairly. After the selection process, each pair of solutions apply crossover using fitness based operator. Finally, the mutation operator is applied using re-balanced operator in order to keep the scheduling balanced.

6.2.2 Ant System Parameters

The parameter values for AS algorithm in this study are adopted from Dorigo and Stutzle (2004). Table 6.5 provides the parameter values for AS algorithm.

Table 6.5

AS parameter value

Parameter	Value
Iteration No	3000
Alpha	1
Beta	8
Evaporation rate	0.5
No of ants	No of machines
Time limit	90 seconds

In Table 6.5, the parameter iteration number is the algorithm termination condition. In addition, the time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. The alpha parameter represents the pheromone influence, increasing the alpha value which will influence the ants to rely more on pheromones instead of heuristic value and vice versa. Beta parameter has the influence on heuristic value. Increasing the beta value will force the ants to follow the heuristic values strongly, while reducing its value will influence the ants more towards pheromone value. The duration of pheromone value is determined by the evaporation rate parameter. Evaporation value of one refers to all the pheromone will be evaporated after each cycle. In opposite, if the evaporation value is zero, then no evaporation will occur at all. In AS algorithm, each ant constructs its own solution, whereby the total number of ants is specified by the number of ants' parameter.

6.2.3 Ant Colony System Parameters

ACS algorithm parameter values are adopted from Dorigo and Stutzle (2004). Table 6.6 shows the parameter values.

Table 6.6

ACS parameter values

Parameter	Value
Iteration No	3000
Beta	8
Evaporation rate	0.6
No of ants	10
Exploitation	0.9
Time limit	90 seconds

In Table 6.6, the parameter iteration number, beta, evaporation rate, number of ants, and time limit are defined the same as in ant system algorithm. The extra parameter in ACS algorithm is the exploitation rate parameter. This parameter controls the algorithm exploitation/exploration behaviour. The value one means that the algorithm will do exploitation search in a greedy way, while the value 0 means the ants will search the space randomly.

6.2.4 Tabu Search Parameters

This study adopted the values for tabu search algorithm parameters from Xhafa, Carretero, et al. (2009). Table 6.7 shows the parameter values for TS algorithm.

Table 6.7

TS parameter values

Parameter	Value
Iteration No	3000
Search neighbour by	Transfer, Swap adjacent, and Swap by load

Tabu size	40
Idle iteration	2
Time limit	90 seconds

In Table 6.7, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. One of the most important parameters in tabu search is searching the neighbourhood of the current solution. There are many techniques to perform this search. Three methods are used in this study to search the neighbourhood. First, transfer method works based on transferring between different jobs and machines. Second, swap adjacent works based on interchanging the adjacent jobs in the solution vector. Third, swap by load will interchange between high and low load machines. The tabu size parameter will specify how many moves to keep in the list as a tabu move. TS algorithm will change from soft diversification into strong diversification based on the idle iteration parameter.

6.2.5 BABC, EBABC1, and EBABC2 Parameters

These algorithm implementations and parameter values are adopted from Kim et al. (2013). Table 6.8 reports the parameter value for each algorithm.

Table 6.8

BABC, EBABC1, EBABC2 parameter values

Parameter name	BABC	EBABC1	EBABC2
Iteration No	3000	3000	3000
Number of food source	20	20	20
Limit	800	800	800
Time limit (seconds)	90	90	90
No of FRS	NA	3	NA
Alpha	NA	NA	0.999

NA: Not applicable for specific algorithm(s).

In Table 6.8, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. The number of food source parameter specifies the number of solutions which is equal to the number of employed bees. The parameter limit controls the number of trials to improve the solution. After reaching the limit, the source food is abandoned and the employed bee for that food source becomes a scout. The extra parameters are number of FRS and alpha as proposed by Kim et al. (2013). FRS value is used to incorporate a Flexible Ranking Strategy (FRS) to improve the balance between exploration and exploitation. Alpha is a real number parameter greater than 0 and less than 1.

6.2.6 PSO-GELS Parameters

The parameters values for GELS-PSO are adopted from the study proposed by Pooranian et al. (2013) as shown in Table 6.9.

Table 6.9

PSO-GELS Algorithm Parameters Values

Parameter	Value
Iteration No	3000
Particle No	50
V_max	40
C1	2
C2	2
GC	6.672
Radius	10

In Table 6.9, the parameter iteration number is the algorithm termination condition. In addition, time limit will terminate the algorithm if the specified time elapses before it reaches the total number of iterations. One of the most important parameters in PSO-

GELS is the particle number which represents the number of solutions. The velocity values are initialized with maximum value determined by V-max parameters. C1 and C2 are positive acceleration constants which control the influence of the best and neighbour solutions on the search process (Izakian et al., 2010). GC is a constant with the value 6.672 and Radius is the neighbour radius between the two responses in the search space.

6.2.7 AS(TS), AS+TS, ACS(TS), and ACS+TS Parameters

Based on the stand-alone version of these algorithms, the hybrid approaches utilized the same parameter values as shown in Table 6.10.

Table 6.10

AS, ACS, and TS Algorithms Parameter Values

Parameter name	AS	ACS	TS
Iteration No	3000	3000	3000
Alpha	1	1	NA
Beta	8	8	NA
Evaporation	0.5	0.6	NA
No of ants	No of machines	10	NA
Exploitation	NA	0.9	NA
Time limit (seconds)	45	45	45
Search neighbour	NA	NA	Transfer, Swap adjacent, and Swap by load
Tabu size	NA	NA	40
Idle iteration	NA	NA	2

NA: Not applicable for specific algorithm.

6.2.8 AS(GA), AS+GA, ACS(GA), and ACS+GA Parameters

These hybrid approaches adopted their parameters values from the stand-alone versions of them. Table 6.11 and 6.12 summarize the parameter values.

Table 6.11

AS(GA) and AS+GA Algorithms Parameter Values

AS		GA	
Parameter name	**Value**	**Parameter name**	**Value**
Iteration No	3000	Iteration No	3000
Alpha	1	Time limit	45 seconds
Beta	8	Population size	10
Evaporation	0.5	Intermediate size	6
No of ants	Number of machines	Selection operator	Tournament 3
Time limit	45 seconds	Crossover rate	0.9
		Mutation rate	0.4
		Crossover operator	Fitness based
		Mutation operator	Re-balance

Table 6.12

ACS(GA) and ACS+GA Algorithms Parameter Values

ACS		GA	
Parameter name	**Value**	**Parameter name**	**Value**
Iteration No	3000	Iteration No	3000
Alpha	1	Time limit	45 seconds
Beta	8	Population size	10
Evaporation	0.7	Intermediate size	6
No of ants	10	Selection operator	Tournament 3
Exploration rate	0.9	Crossover rate	0.9
Time limit	45 seconds	Mutation rate	0.4
		Crossover operator	Fitness based
		Mutation operator	Re-balance

The execution time for GA, AS, ACS, TS, BABC, EBABC1, EBABC2 is set to 90 seconds which is commonly used as a reasonable amount of time for scheduling jobs in a computational grid environment (Xhafa, Barolli, et al., 2007a; Xhafa & Duran, 2008). To keep this restriction time in the hybrid approaches, each algorithm is limited to 45 seconds in order to conduct fair experiments. However, each algorithm is also terminated with an iteration number which also works as a termination

criterion besides the limited time. In other words, each algorithm is terminated either by reaching the allowed run time or reaching the allowed iteration number.

6.3 Experimental Result and Analysis

The experiments are conducted using Intel® Core (TM) i7-3612QM CPU @ 2.10 GHz and 8G RAM. The grid computing simulator is developed using visual C#. The time given for each experiment is 90 seconds (45 seconds for each algorithm in the hybrid approach). Each algorithm is executed 10 times in order to calculate the best and average values. The proposed algorithms are evaluated based on makespan, flowtime, and utilization metrics.

6.3.1 Best Makespan Results

In order to compare and represent the performance of the proposed algorithms visually, a geometric mean is calculated to normalize the makespan, flowtime, and utilization values of five instances (Izakian, Abraham, & Snsel, 2009). In addition, the difference between ACS algorithm and the proposed hybrid algorithm is calculated to provide the enhancement of each algorithm in terms of percentage.

Figure 6.1 displays the results of the sixteen algorithms in terms of the best makespan value. The Figure 6.1 shows that the worse performance produced by BABC, EBABC2 and EBABC1algorithms. The algorithms, AS, AS(GA), and AS(TS) show similar performance to each other which is better than BABC, EBABC2 and EBABC1 algorithms. The performance enhanced slightly by AS+TS, AS+GA, PSO-GELS, GA, ACS, ACS(GA), ACS+GA, TS, and ACS+TS algorithms. The proposed hybrid algorithm ACS(TS) achieved the best performance in terms of best makespan

value as shown in Figure 6.1. This is due to the enhancement process achieved by TS algorithm to the solution produced by each cycle in ACS algorithm.

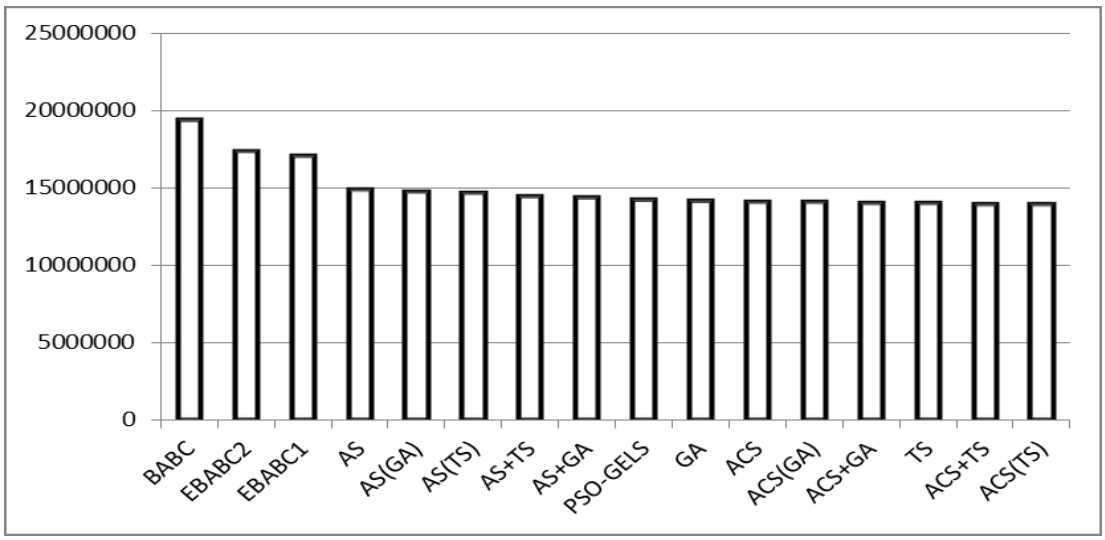

Figure 6.1. Geometric mean for the best makespan values

Figure 6.2 represents the enhancement of each hybrid algorithm which is expressed in terms of percentage. Each hybrid algorithm is compared with the ACS algorithm in terms of best makespan value enhancement. The Figure 6.2 shows that ACS(TS) algorithm enhanced 1.26% followed by ACS+TS 1.16%, ACS+GA 0.56%, and ACS(GA) 0.01%. This enhancement indicates that GA and TS algorithms increased the performance of ant colony system algorithm in both low and high levels. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in low level order. Clearly in low level hybridization between ACS and TS, the TS algorithm enhances the best solution at the end of each cycle of ACS algorithm. This makes the ants update the pheromone based on the enhanced solution produced from TS algorithm. Therefore, the low level hybridization algorithm was able to achieve good results.

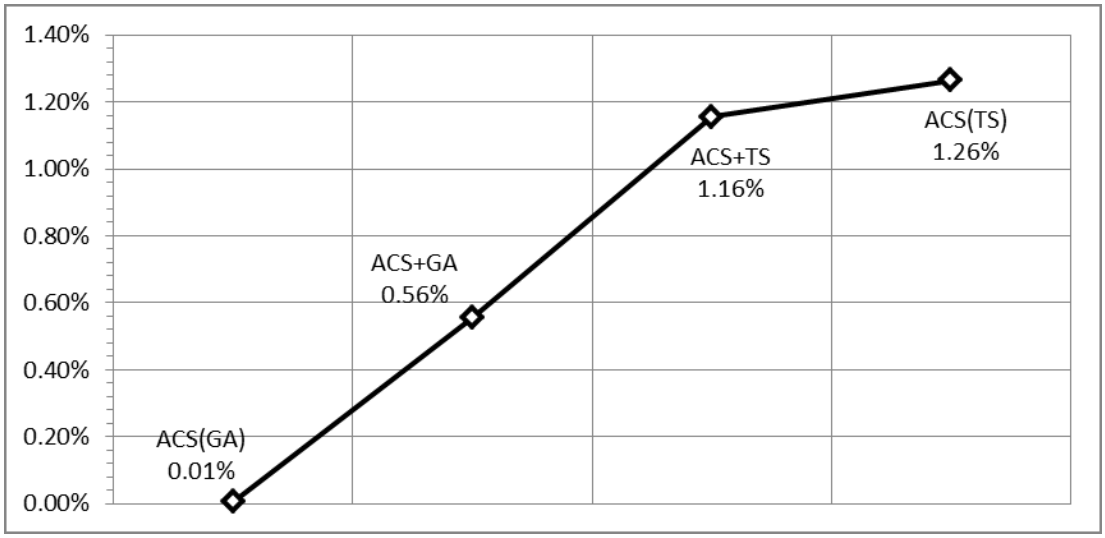

Figure 6.2. The percentage enhancement of each hybrid algorithm in terms of the best makespan values

6.3.2 Average Makespan Results

Figure 6.3 displays the results of the sixteen algorithms in terms of the average makespan value. The Figure 6.3 shows that the worse performance produced by BABC followed by EBABC2 and EBABC1 algorithms. The algorithms AS, AS(GA), and AS(TS) show similar performance to each other which is better than BABC, EBABC2 and EBABC1 algorithms. The performance enhanced slightly by AS+TS, AS+GA, ACS, PSO-GELS, ACS(GA), GA, ACS+GA, TS, and ACS+TS algorithms. The proposed hybrid algorithm ACS(TS) achieved the best performance in terms of average makespan value as shown in Figure 6.3. This is due to the enhancement process achieved by TS algorithm to the solution produced by each cycle in ACS algorithm.

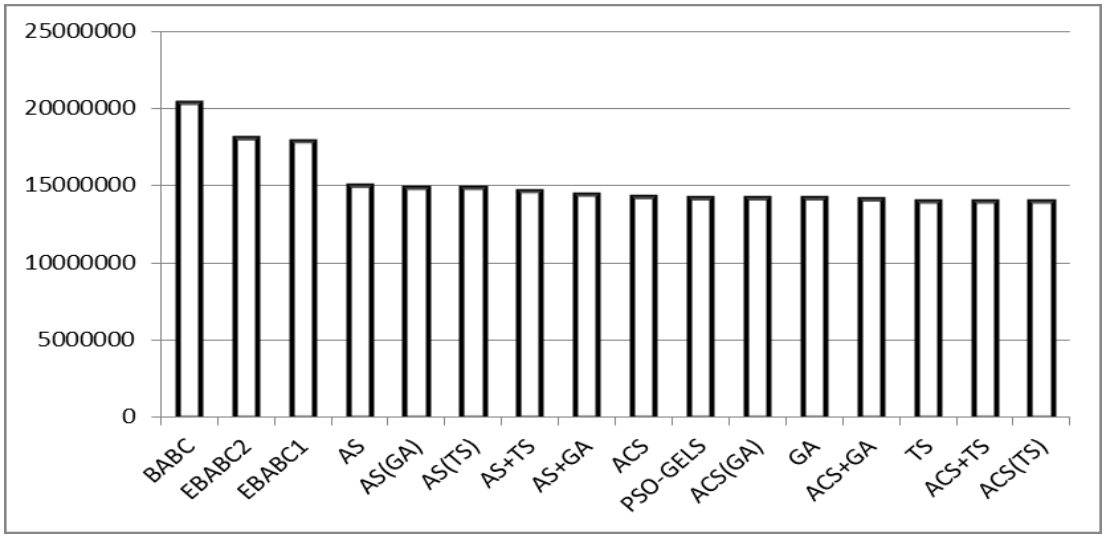

Figure 6.3. Geometric mean for the average makespan values

Figure 6.4 shows that ACS(TS) algorithm enhanced 1.96% followed by ACS+TS 1.85%, ACS+GA 1.01%, and ACS(GA) 0.43%. This enhancement indicates that GA and TS algorithms increased the performance of ant colony system algorithm in both low and high levels. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in low level order.

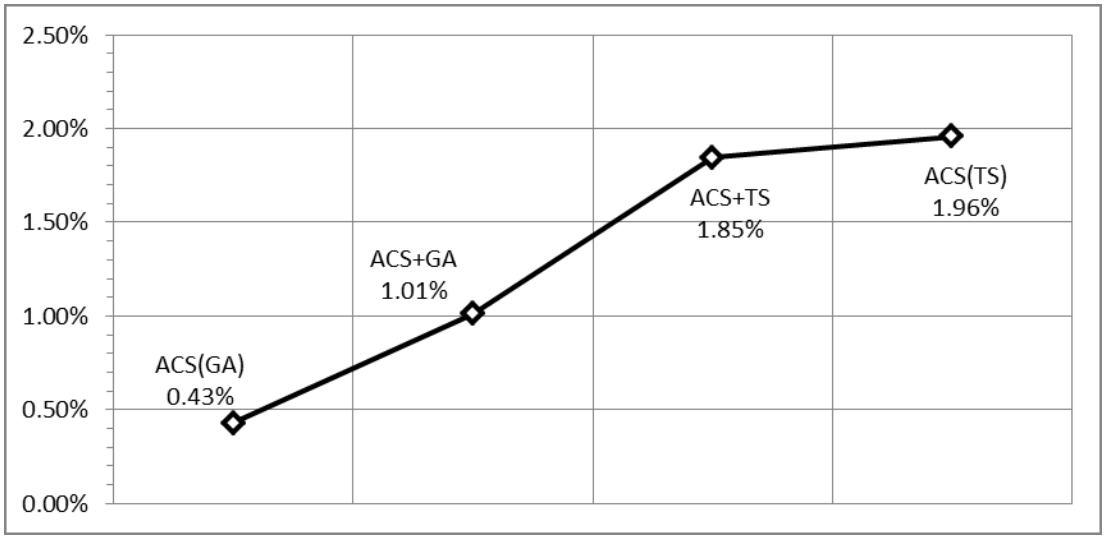

Figure 6.4. The percentage enhancement of each hybrid algorithm in terms of the average makespan values

6.3.3 Best Flowtime Results

The second metric to measure the scheduling algorithm performance is the flowtime. Figure 6.5 displays the results of the sixteen algorithms in terms of the best flowtime value. The Figure 6.5 shows that the worse performance produced by BABC followed by EBABC1 algorithms. The algorithms AS(GA), AS(TS), AS+GA, EBABC2, AS+TS, and AS show similar performance to each other which is better than BABC and EBABC1 algorithms. The performance enhanced slightly with ACS(GA), ACS, PSO-GELS, GA, ACS+GA, TS, and ACS(TS) algorithms. The proposed hybrid algorithm ACS+TS achieved the best performance in terms of best flowtime value as shown in Figure 6.5. This is due to the refinement process achieved by TS algorithm to the best solution produced by ACS algorithm.

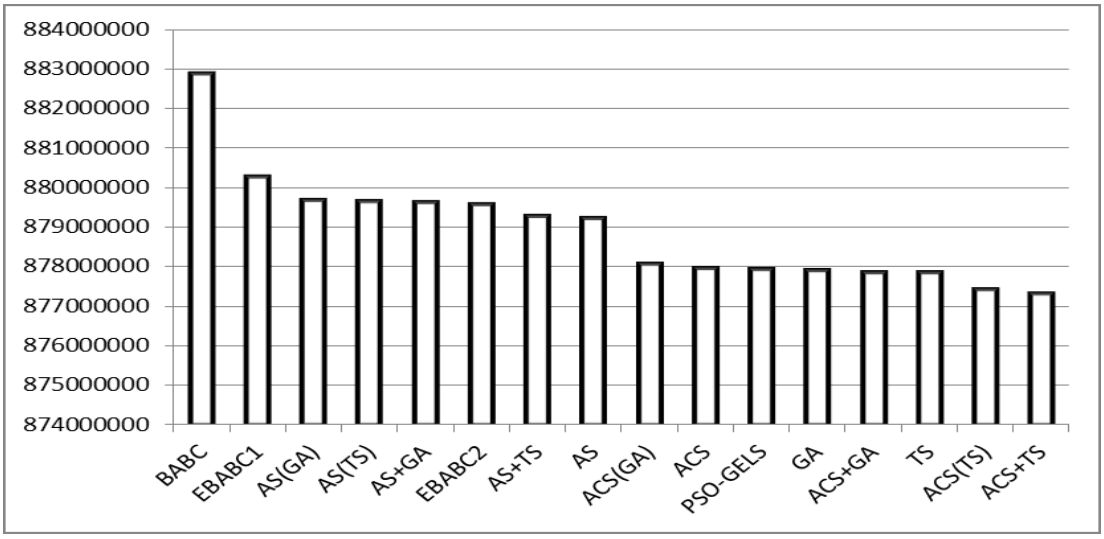

Figure 6.5. Geometric mean for the best flowtime values

Figure 6.6 shows that ACS+TS algorithm enhanced 0.07% followed by ACS(TS) 0.06%, and ACS+GA 0.01%. The proposed hybrid algorithm ACS(GA) perform worse than ACS algorithm (not included in the graph).

This enhancement indicates that TS algorithm increased the performance of ant colony system algorithm in both low and high levels while GA algorithm only increases the ACS algorithm performance in high level order. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in high level order. Clearly in high level order, the TS algorithm starts with initial solution passed from ACS algorithm which is very high quality produced by the best ant. Therefore, TS starts from good location in the search space which leads to further enhancement to the solution found by ACS algorithm.

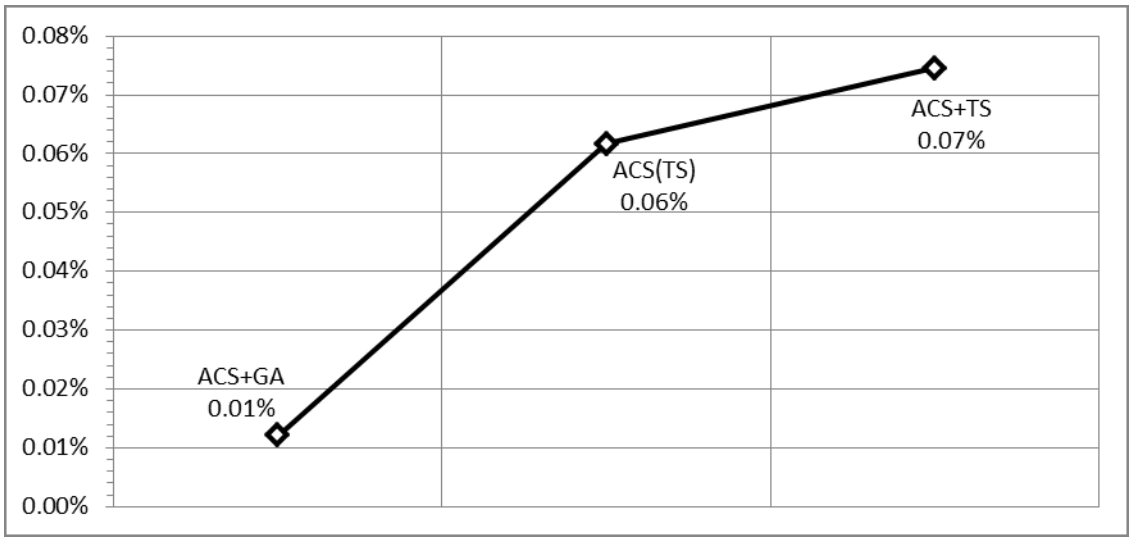

Figure 6.6. The percentage enhancement of each algorithm in terms of the best flowtime values

6.3.4 Average Flowtime Results

Figure 6.7 displays the results of the sixteen algorithms in terms of the average flowtime value. The Figure 6.7 shows that the worse performance achieved by BABC followed by EBABC1, and EBABC2 algorithms. The algorithms AS(GA), AS(TS), AS+TS, AS+GA, and AS show similar performance to each other which is better than BABC, EBABC1, and EBABC2 algorithms. The performance enhanced little with ACS(GA), ACS, ACS+GA, PSO-GELS, GA, TS, and ACS+TS algorithms. The

proposed hybrid algorithm ACS(TS) achieved the best value in terms of the average flowtime values as shown in Figure 6.7. This is due to the enhancement process achieved by TS algorithm to the solution produced by each cycle in ACS algorithm.

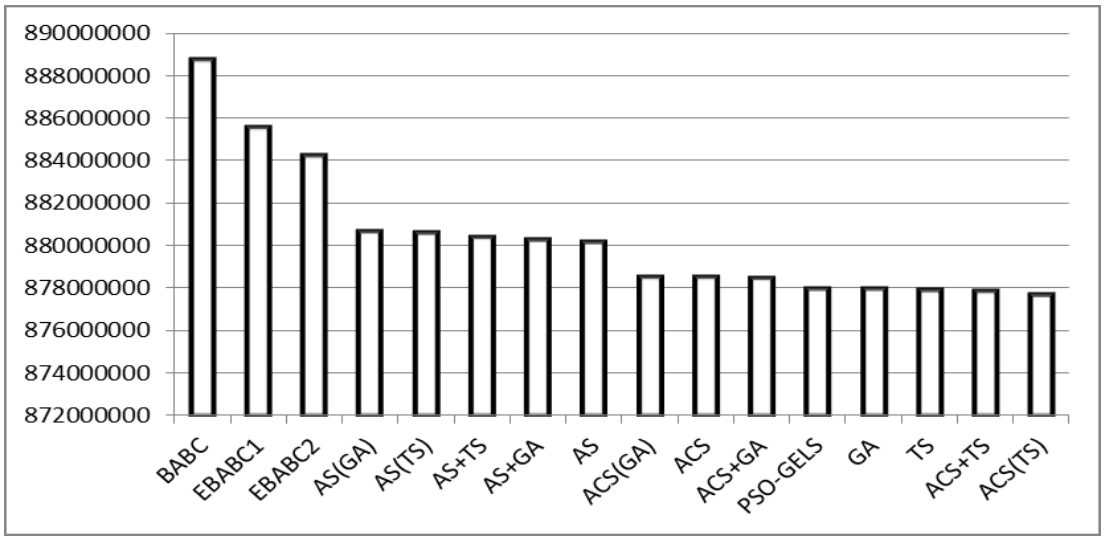

Figure 6.7. Geometric mean for the average flowtime values

Figure 6.8 shows that ACS(TS) algorithm enhanced 0.093% followed by ACS+TS 0.076%, and ACS+GA 0.001%. The proposed hybrid algorithm ACS(GA) perform worse than ACS algorithm (not included in the graph).

This enhancement indicates that TS algorithm increased the performance of ant colony system algorithm in both low and high levels while GA algorithm only increases the ACS algorithm performance in high level order. However, the best performance achieved when ACS algorithm hybridized with TS algorithm in low level order.

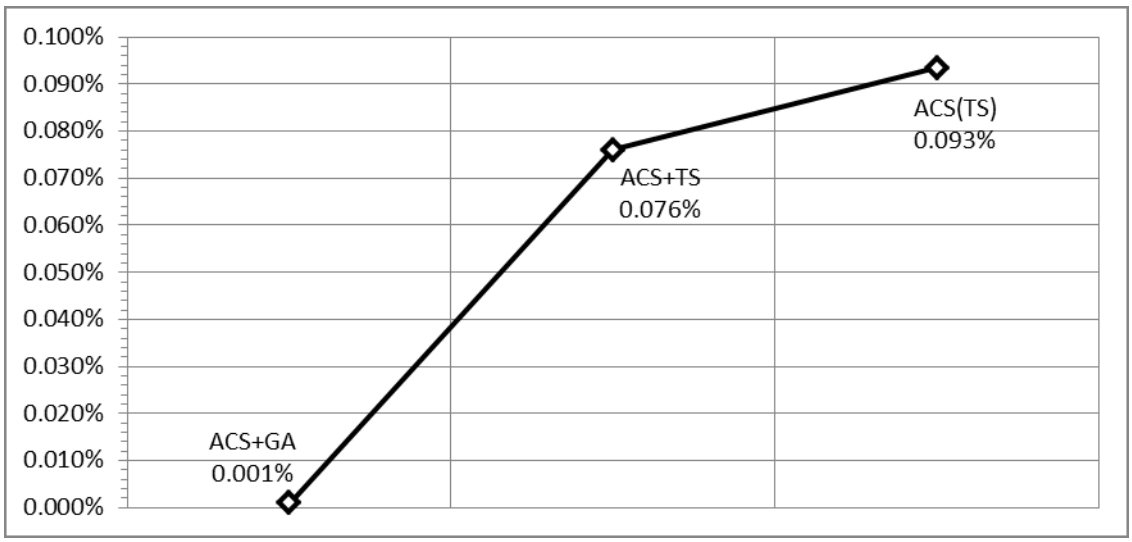

Figure 6.8. The percentage enhancement of each hybrid algorithm in terms of the average flowtime values

6.3.5 Best Utilization Results

The last metric implemented in this study to measure the algorithms performance is the utilization of resources in the computational grid. Figure 6.9 displays the results of the sixteen algorithms in terms of the best utilization value. The Figure 6.9 shows that the worse performance achieved by BABC, EBABC2, and EBABC1 algorithms. The algorithms AS, AS(TS), AS+TS, AS+GA, AS(GA), PSO-GELS, GA, and ACS(GA) show similar performance to each other which is better than BABC, EBABC2, and EBABC1 algorithms. The performance enhanced little with ACS, ACS+GA, TS, and ACS+TS algorithms. The proposed hybrid algorithm ACS(TS) achieved the best value in terms of best utilization as shown in Figure 6.9. However, the algorithms ACS+GA, TS, and ACS+TS show good performance in terms of best utilization value.

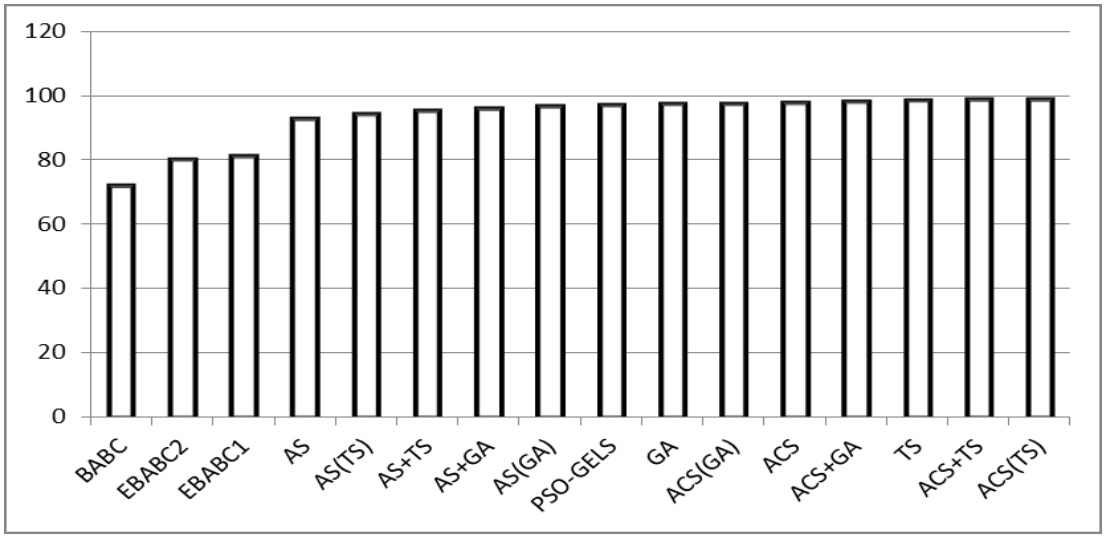

Figure 6.9. Geometric mean for the best utilization value

Figure 6.10 shows that ACS(TS) algorithm enhanced over ACS algorithm by 1.19% followed by ACS+TS enhanced by 1.09% and finally ACS+GA algorithm enhanced over ACS algorithm by 0.52%. The low level hybridization between ACS and GA did not enhance the ACS algorithm.

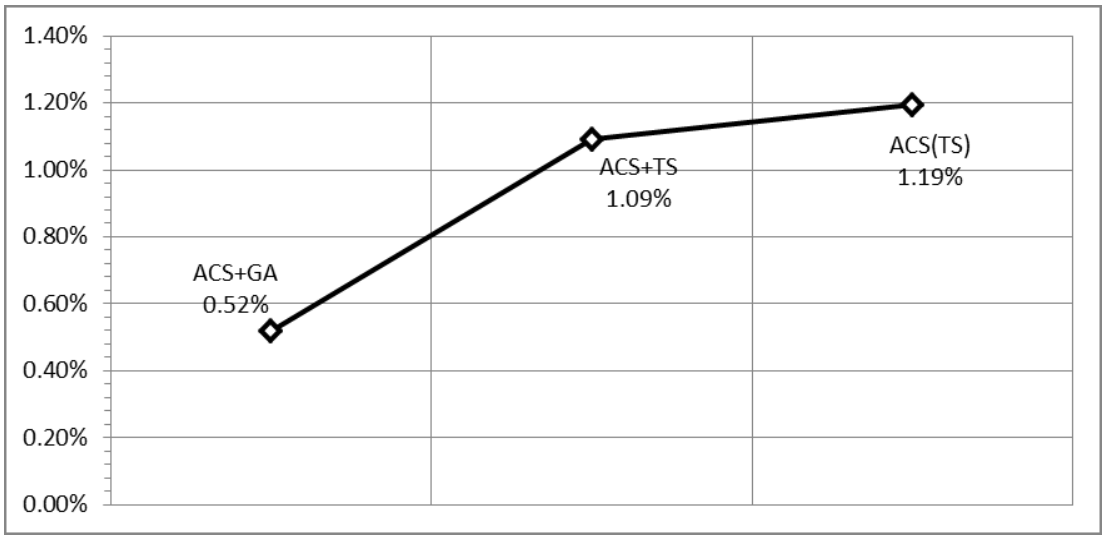

Figure 6.10. The percentage enhancement of each hybrid algorithm in terms of the best utilization values

6.3.6 Average Utilization Results

Figure 6.11 displays the results of the sixteen algorithms in terms of the average utilization value. The Figure 6.11 shows that the best performance achieved by the proposed hybrid ACS(TS) algorithm. Similar performance also achieved by ACS+TS, TS, ACS+GA, GA, PSO-GELS, ACS(GA), and ACS algorithms. Slightly less performance achieved by AS+GA, AS(GA), AS+TS, AS(TS), and AS algorithms. The worse performance achieved by EBABC1, EBABC2, and BABC algorithms.

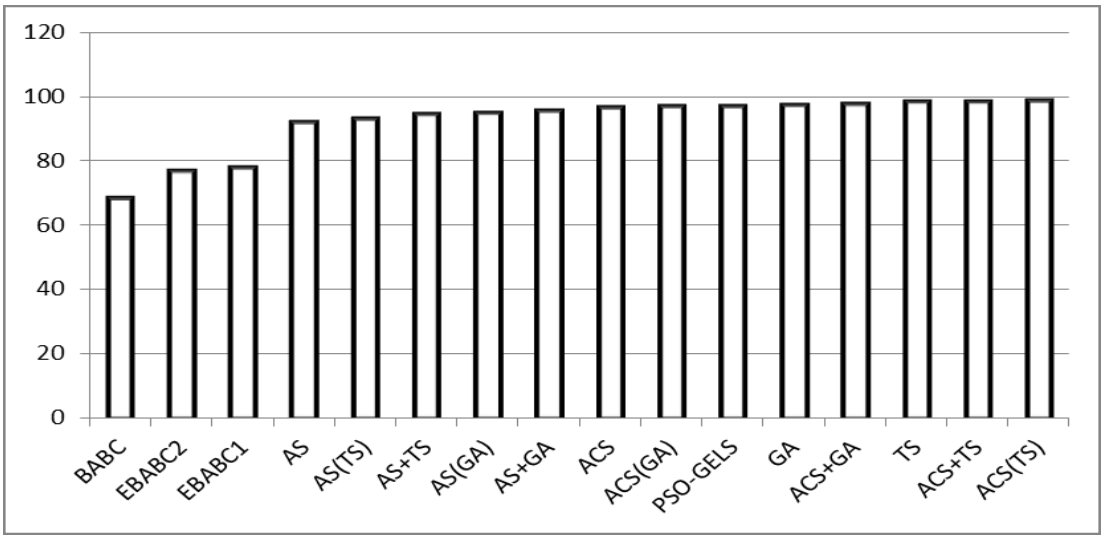

Figure 6.11. Geometric mean for the average utilization value

Figure 6.12 shows that in terms of the average utilization value, ACS(TS) algorithm enhanced over ACS algorithm by 1.9% and ACS+TS enhanced over ACS algorithm by 1.8%. In addition, ACS+GA algorithm enhanced over ACS algorithm by 1.01% and ACS(GA) algorithm enhanced over ACS algorithm by 0.17%.

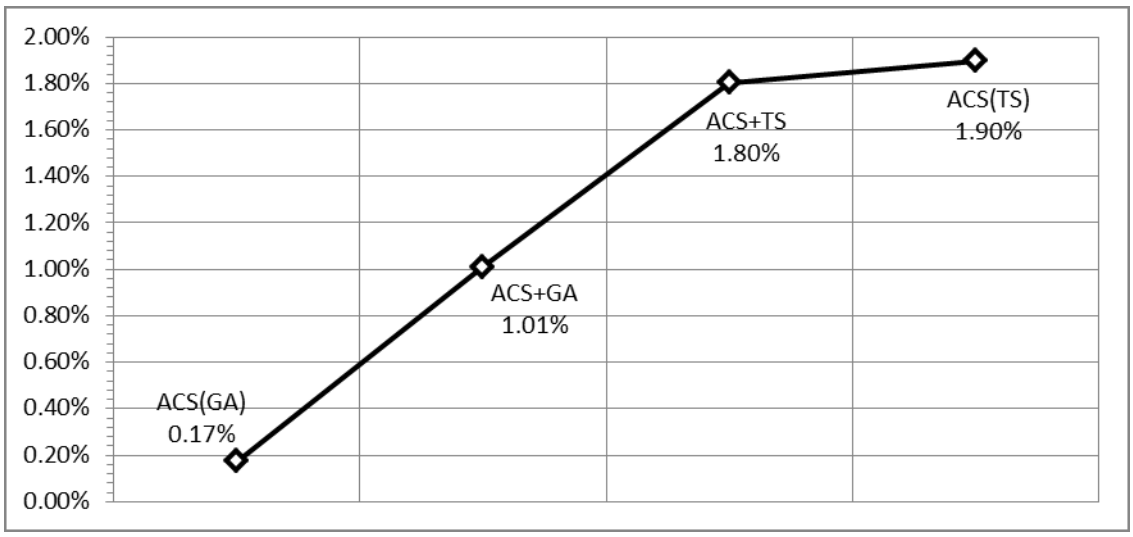

Figure 6.12. The percentage enhancement of each hybrid algorithm in terms of the average utilization values

6.3.7 Discussion

In terms of best makespan value, the proposed algorithm ACS(TS) outperform the other algorithms. The best makespan value is the main objective of this study which reflects the productivity of grid computing system. ACS(TS) achieved good results due to the ability of TS algorithm to enhance the solution produced by the ants at the end of the each iteration in ACS algorithm. In low level hybridization ACS(TS), ACS algorithm uses the solution produced by TS for global pheromone update. Therefore, the ants in the nest iteration will be influenced by the solution enhanced by TS algorithm.

The worse performance produced by BABC algorithm. BABC is the binary version of ABC algorithm which originally developed to solve continues problems. In spite of several studies proposed methods to apply ABC algorithm into discrete problems, the difficulty there is no natural extensions for combinatorial problems, such as travelling salesman problem and scheduled problems (Poli et al., 2007).

6.4 Summary

From these findings, it is clear that ACS algorithm did not produce good solutions when it is executed alone. On the other hand, the local search algorithms GA and TS performances depend on the initial solution. Hence, GA and TS algorithm performances are enhanced when they are hybridized with ACS. Therefore, the solution produced by ACS needs to be refined using local search which is done successfully using the proposed algorithms in this study, specifically ACS(TS) algorithm. It is important to mention that the execution time given is 90 seconds in order to provide the real requirements in grid computing environment. Such a strong time concentration makes the algorithm terminate before it finishes its iteration number.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

Similar to any other NP-complete problems, job scheduling in grid computing is a real challenging problem which is very hard to tackle. These types of problems required metaheuristics algorithms to achieve near optimal solution. Each metaheuristics algorithm shows good performance in specific problem. Therefore, no algorithm could be said is good for all NP-complete problems. Moreover, some algorithms are able to achieve good results only in specific instances of the specific problem. Due to this restriction, this study has been focused on enhancing ant colony system for job scheduling problem in grid computing. This study has been implemented a hybrid approach between ACS, GA, and TS algorithms. Section 7.1 discusses the research contribution while the research limitations are presented in Section 7.2. Suggested future work is provided in Section 7.3.

7.1 Research Contribution

One way to enhance ant colony system is by hybridizing it with other algorithms. In this study, it is found that genetic algorithm and tabu search algorithm are very useful to enhance ACS algorithm for job scheduling in grid computing. This study has five contributions specifically, four hybrid algorithms and one simulator.

The first contribution is the low level hybridization between ant colony system and genetic algorithm which is called ACS(GA) algorithm. This hybrid algorithm has the ability to enhance the exploration mechanism in ACS algorithm. In terms of best makespan value for the static environment, ACS(GA) enhanced over ACS algorithm

by 0.35%. For dynamic environment, ACS(GA) enhanced over ACS algorithm by 0.01%.

The second contribution is the low level hybridization between ant colony system and tabu search algorithm which is called ACS(TS) algorithm. This hybrid algorithm has the ability to also enhance the exploration mechanism in ACS algorithm. In terms of best makespan value for the static environment, ACS(TS) enhanced over ACS algorithm by 4.65%. For dynamic environment, ACS(TS) enhanced over ACS algorithm by 1.26%.

The third contribution is the high level hybridization between ant colony system and genetic algorithm which is called ACS+GA algorithm. This hybrid algorithm has the ability to refine the solution produced by ACS algorithm. In terms of best makespan value for the static environment, ACS+GA enhanced over ACS algorithm by 2.03%. For dynamic environment, ACS+GA enhanced over ACS algorithm by 0.56%.

The fourth contribution is the high level hybridization between ant colony system and tabu search algorithm which is called ACS+TS algorithm. This hybrid algorithm has the ability to refine the solution produced by ACS algorithm. In terms of best makespan value for the static environment, ACS+TS enhanced over ACS algorithm by 6.99%. For dynamic environment, ACS+TS enhanced over ACS algorithm by 1.16%.

The fifth contribution is the grid computing simulator called ExSim simulator which has the ability to simulate static and dynamic environments for job scheduling. A trial version of ExSim simulator could be requested from http://exsim.webs.com/.

The experimental results found that the hybridization between ACS and TS, specifically the high level hybridization ACS+TS achieved the best performance compared to other algorithms in static grid computing environment. Therefore, this study recommends applying ACS+TS algorithm for a grid computing system which has the characteristics similar to static environment. In addition, the study recommends applying ACS(TS) algorithm for dynamic computational grid system.

7.2 Limitation of the Study

Limitations are unavoidable events in any study and this study has no exceptions. The followings are the limitations of this study:

i. For algorithm evaluation purpose, there is no standard simulator that could be used, especially when the required algorithms are hybridized like the proposed algorithms in this study. Therefore, a simulator is developed from scratch in order to conduct the evaluation experiments.

ii. Due to the random variables used in the distribution methods to generate the benchmark problems, it is not possible to compare results reported in the literature. Hence, all the algorithms which were selected for comparison were implemented in this study.

iii. This study did not consider the variation of the network connection speed.

7.3 Recommendation for Future Work

During the process of this study, several directions arose which are considered as good seeds for future research. The following points highlight the promising directions found by this study:

i. There are many other components in grid computing systems which need to be investigated, such as security, task immigration between resources, and the resource fault predicting.

ii. Grid computing field needs standardization in terms of infrastructure and implementation in order to unify the grid architecture.

iii. The enhanced scheduling algorithm could be investigated with other scheduling problems, such as job shop scheduling problem and Flow shop scheduling problem.

iv. Other local search algorithms could be hybridized with ACS algorithms, such as simulated annealing and artificial bee colony.

v. One of the most important components in tabu search algorithm is the neighbourhood search techniques. There are several techniques which could be investigated in order to find the best combination for tabu search algorithm.

vi. During the iterations of ant colony system algorithm, after each cycle, each ant will calculate the fitness of its solution and compare it with the best-so-far solution. The process of calculating the solution fitness is time consuming. On the other hand, these solutions could already be found and calculated in some previous loops or by other ants which will waste time in calculating the same solution again. Therefore, introducing a hash function to convert the solution into hash code and save it in a hash table will reduce the calculation time. Each ant will use the hash function to convert its solution into a hash code and compare it with the codes saved in the table. If the code already exists in the table, then there is no need to calculate its fitness value and simply discard the solution. Otherwise, save the new code in the hash table and calculate the fitness value. This enhancement will save a lot of time in ACS run time.

REFERENCES

Aarts, E., Korst, J., & Michiels, W. (2014). Simulated Annealing. In E. K. Burke & G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 265–285). Boston: Springer.

Aarts, E., & Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization*. Princetonn: Princeton University Press.

Abraham, A., Grosan, C., & Ishibuchi, H. (2007). *Hybrid Evolutionary Algorithms*. Heidelberg: Springer.

Abraham, A., Liu, H., Zhang, W., & Chang, T. (2006). Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm. In *Proceedings of the 10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems* (pp. 500–507). Bournemouth.

Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli, D., & Soukhal, A. (2014). *Multiagent Scheduling Models and Algorithms*. Heidelberg: Springe.

Alba, E., Almeida, F., Blesa, M., Cabeza, J., Cotta, C., Díaz, M., … Xhafa, F. (2002). MALLBA: A Library of Skeletons for Combinatorial Optimisation. In *Proceedings of the 8th International Euro-Par Conference on Parallel Processing* (pp. 927–932). Paderborn.

Alba, E., Almeida, F., Blesa, M., Cotta, C., Diaz, M., Dorta, I., … Xhava, F. (2006). Efficient Parallel LAN / WAN Algorithms for Optimization . The MALLBA Project. *Journal of Parallel Computing*, *32*(5-6), 415–440.

Aleti, A. (2012). *An Adaptive Approach to Controlling Parameters of Evolutionary Algorithms*. (Doctoral dissertation). Retrieved from http://researchbank.swinburne.edu.au/vital/access/manager/Index

AL-Fawair, M. A. (2009). *A Framework for Evolving Grid Computing Systems*. (Doctoral dissertation). Retrieved from http://www.diva-portal.org/smash/search.jsf

Ali, S. S., Siegel, H. J., Maheswaran, M., Hensgen, D., & Lafayette, W. (2000). Task Execution Time Modeling for Heterogeneous Computing Systems. In *Proceedings of the 9th Heterogeneous Computing Workshop* (pp. 185–199). Cancun. doi:10.1109/HCW.2000.843743

Ali, S., Siegel, H. J., Maheswaran, M., Hensgen, D., & Ali, S. (2000). Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems. *Tamkang Journal of Science and Engineering*, *3*(3), 195–207.

Anousha, S., Anousha, S., & Ahmadi, M. (2014). A New Heuristic Algorithm for Improving Total Completion Time in Grid Computing. In James J. Park, S.-C.

Chen, J.-M. Gil, & N. Y. Yen (Eds.), *Multimedia and Ubiquitous Engineering* (pp. 17–26). Heidelberg: Springe.

Aron, R., & Chana, I. (2012). Formal QoS Policy Based Grid Resource Provisioning Framework. *Journal of Grid Computing*, *10*(2), 249–264. doi:10.1007/s10723-012-9202-y

Atasagun, Y., & Kara, Y. (2014). Bacterial Foraging Optimization Algorithm for Assembly Line Balancing. *Journal of Neural Computing and Applications*, *25*(1), 237–250.

Babafemi, O., Sanjay, M., & Adigun, M. (2013). Towards Developing Grid-Based Portals for E-Commerce on-Demand Services on a Utility Computing Platform. *Journal of Procedia*, *4*(1), 81–87. doi:10.1016/j.ieri.2013.11.013

Bagherzadeh, J., & MadadyarAdeh, M. (2009). An Improved Ant Algorithm for Grid Scheduling Problem. In *Proceedings of the 14th International Conference on CSI Computer* (pp. 323–328). Tehran.

Bai, J., Chen, L., Jin, H., Chen, R., & Mao, H. (2012). Robot Path Planning Based on Random Expansion of Ant Colony Optimization. In Z. C. Qian, W. Su, T. Wang, & H. Yang (Eds.), *Recent Advances in Computer Science and Information Engineering* (pp. 141–146). Heidelberg: Springer.

Bai, L., Hu, Y., Lao, S., & Zhang, W. (2010). Task scheduling with load balancing using multiple ant colonies optimization in grid computing. *2010 Sixth International Conference on Natural Computation*, (Icnc), 2715–2719. doi:10.1109/ICNC.2010.5582599

Bandieramonte, M., Stefano, A. Di, & Morana, G. (2008). An ACO Inspired Strategy to Improve Jobs Scheduling in a Grid Environment. In *Proceedings of the 8th International Conference on Algorithms and Architectures for Parallel Processing* (pp. 30–41). Cyprus.

Bardsiri, A. K., & Hashemi, S. M. (2012). A Comparative Study on Seven Static Mapping Heuristics for Grid Scheduling Problem. *International Journal of Software Engineering and Its Applications*, *6*(4), 247–256.

Berman, F., Fox, G., & Hey, A. J. G. (2003). *Grid Computing: Making the Global Infrastructure a Reality*. Chichester, England: Wiley.

Birattari, M. (2009). *Tuning Metaheuristics A Machine Learning Perspective*. Berlin: Springer.

Biswal, B., Dash, P. K., & Mishra, S. (2011). A Hybrid Ant Colony Optimization Technique for Power Signal Pattern Classification. *Journal of Expert Systems with Applications*, *38*(5), 6368–6375.

Blum, C., & Li, X. (2008). Swarm Intelligence in Optimization. In C. Blum & D. Merkle (Eds.), *Swarm Intelligence*. Heidelberg: Springer.

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *Journal of ACM Computing Surveys*, *35*(3), 268–308. doi:10.1145/937503.937505

Botzheim, J., Toda, Y., & Kubota, N. (2012). Bacterial Memetic Algorithm for Simultaneous Optimization of Path Planning and Flow Shop Scheduling Problems. *Journal of Artificial Life and Robotics*, *17*(1), 107–112.

Boussaid, I., Lepagnot, J., & Siarry, P. (2013). A Survey on Optimization Metaheuristics. *Journal of Information Sciences*, *237*(1), 82–117.

Brade, D., & Lehmann, A. (2002). Model Verification and Validation. In A. N. Ince (Ed.), *Modeling and Simulation Environment for Satellite and Terrestrial Communications Networks*. Boston: Springer. doi:10.1007/978-1-4615-0863-2_17

Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., … Freund, R. F. (1999). A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems. In *Proceedings of the 8th Heterogeneous Computing Workshop* (pp. 15–29). San Juan.

Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., … Freund, R. F. (2001). A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, *61*(6), 810–837. doi:10.1006/jpdc.2000.1714

Bullnheimer, B., Hart, R. F., & Straub, C. (1999). A New Rank-Based Version of the Ant System: A Computational Study. *Central European Journal of Operations Research and Economics*, *7*(1), 25 – 38.

Burke, E. K., & Kendall, G. (2014). *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. New York: Springer.

Buyya, R., & Murshed, M. (2002). GridSim: a Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Journal of Concurrency and Computation: Practice and Experience*, *14*(13), 1175–1220. doi:10.1002/cpe.710

Cai, R., Ning, Z., Li, L., & Zhong, Y. (2007). Simulated Annealing Algorithm for Independent Tasks Assignment in Heterogeneous Computing Systems. In *Proceedings of the 3rd International Conference on Natural Computation* (pp. 105–109). Haikou.

Calvete, H. I., Gale, C., & Oliveros, M. (2012). Ant Colony Optimization for Solving the Vehicle Routing Problem with Delivery Preferences. In *Proceedings of the International Conference on Modeling and Simulation in Engineering, Economics and Management* (pp. 230–239). New Rochelle.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraula, G., & Bonabeau, E. (2003). *Self-Organization in Biological Systems*. Princeton, N.J. Oxford: Princeton University Press.

Caron, E., Garonne, V., & Tsaregorodtsev, A. (2007). Definition, Modelling and Simulation of a Grid Computing Scheduling System for High Throughput Computing. *Journal of Future Generation Computer Systems*, *23*(8), 968–976. doi:10.1016/j.future.2007.04.008

Carretero, J., & Xhafa, F. (2006). Use of Genetic Algorithms for Scheduling Jobs in Large Scale Grid Applications. *Journal of Technological and Economic Development of Economy*, *12*(1), 11–17.

Carretero, J., Xhafa, F., & Abraham, A. (2007). Genetic Algorithm Based Schedulers for Grid Computing Systems. *International Journal of Innovative Computing, Information and Control*, *3*(6), 1–19.

Carvalho, M., & Brasileiro, F. (2012). A User-Based Model of Grid Computing Workloads. In *Proceedings of the 13th ACM/IEEE International Conference on Grid Computing* (pp. 40–48). Beijing.

Chang, R., Chang, J., & Lin, P.-S. (2009). An Ant Algorithm for Balanced Job Scheduling in Grids. *Journal of Future Generation Computer Systems*, *25*(1), 20–27. doi:10.1016/j.future.2008.06.004

Chang, R.-S., Chang, J.-S., & Lin, P.-S. (2007). Balanced Job Assignment Based on Ant Algorithm for Computing Grids. In *Proceedings of the 2nd IEEE Asia-Pacific Conference on Service Computing* (pp. 291–295). Tsukuba Science City.

Chaturvedi, A. K., & Sahu, R. (2011). New Heuristic for Scheduling of Independent Tasks in Computational Grid. *International Journal of Grid and Distributed Computing*, *4*(3).

Chen, X., Kong, Y., Fang, X., & Wu, Q. (2011). A Fast Two-Stage ACO Algorithm for Robotic Path Planning. *Journal of Neural Computing and Applications*, *22*(2), 313–319.

Chen, Y., Yu, L., Hong, Z., & Dong, Q. (2012). On Energy-Saving Routing Based on Ant Colony Algorithm for Wireless Sensor Networks. In *Proceedings of the International Conference in Electrics, Communication and Automatic Control Proceedings* (pp. 1241–1247). Chongqing.

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. In *Proceedings of the European Conference on Artificial Life* (pp. 134 – 142). Paris.

Conejero, J., Caminero, B., Carrion, C., & Tomas, L. (2014). From Volunteer to Trustable Computing: Providing QoS-Aware Scheduling Mechanisms for Multi-Grid Computing Environments. *Journal of Future Generation Computer Systems*, *34*(1), 76–93. doi:10.1016/j.future.2013.12.005

Cordon, O., Viana, I. F. de, & Herrera, F. (2002). Analysis of the Best-Worst Ant System and its Variants on the TSP. *Journal of Mathware and Soft Computing*, *9*(3), 228–234.

Cordon, O., Viana, I. F. de, Herrera, F., & Moreno, L. (2000). A New ACO Model Integrating Evolutionary Computation Concept: The Best-Worst Ant System. In *Proceedings of the 3rd International Workshop on Ant Algorithms* (pp. 22–29). Granada.

Costa, D. (1994). An Evolutionary Tabu Search Algorithm and the NHL Scheduling Problem. *Journal of INFOR*, *33*(1), 161–178.

Cyril Daisy Christina, P., & Miriam, D. D. H. (2012). Adaptive Task Scheduling Based on Multi Criterion Ant Colony Optimization in Computational Grids. In *Proceedings of the International Conference on Recent Trends in Information Technology* (pp. 185–190). Tamil Nadu.

David, N. (2013). Validating Simulations. In B. Edmonds & R. Meyer (Eds.), *Simulating Social Complexity*. Berlin: Springer. doi:10.1007/978-3-540-93813-2_8

Davis, L. D. (1991). *Handbook Of Genetic Algorithms*. New York: Van Nostrand Reinhold Computer Library.

Devi, S. N., & Pethalakshmi, A. (2012). Resource Discovery for Grid Computing Environment Using Ant Colony Optimization by Applying Routing Information and LRU Policy. In *Proceedings of the 4th International Conference on Global Trends in Computing and Communication Systems* (pp. 124–133). Vellore, India.

Do Duc, D., Dinh, H. Q., Dang, T. H., Laukens, K., & Hoang, X. H. (2012). AcoSeeD: An Ant Colony Optimization for Finding Optimal Spaced Seeds in Biological Sequence Search. In *Proceedings of the 8th International Conference on Swarm Intelligence* (pp. 204–211). Brussels.

Dokeroglu, T., & Cosar, A. (2012). Dynamic Programming with Ant Colony Optimization Metaheuristic for Optimization of Distributed Database Queries. In *Proceedings of the 26th International Symposium on Computer and Information Sciences* (pp. 107–113). London.

Dorigo, M., & Gambardella, L. M. (1997a). Ant Colonies for the Travelling Salesman Problem. *Journal of BioSystems*, *43*(2), 73–81. doi:10.1016/S0303-2647(97)01708-5

Dorigo, M., & Gambardella, L. M. (1997b). Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem. *Journal of IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66. doi:10.1109/4235.585892

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *Positive Feedback as a Search Strategy (Report No 91- 016)* (pp. 1–20). Milan.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant System: Optimization by a Colony of Cooperating Agents. *Journal of IEEE Transactions on Systems, Man, and Cybernetics-Part B, Cybernetics*, *26*(1), 29–41. doi:10.1109/3477.484436

Dorigo, M., & Stutzle, T. (2004). *Ant Colony Optimization*. Cambridge, Mass: MIT Press.

Dorigo, M., Stutzle, T., & Stützle, T. (2003). The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 250–285). Boston: Kluwer Academic Publishers.

Douiri, S. M., & Elbernoussi, S. (2012). A New Ant Colony Optimization Algorithm for the Lower Bound of Sum Coloring Problem. *Journal of Mathematical Modelling and Algorithms*, *11*(2), 181–192.

Dumitrescu, C. L., & Foster, I. (2005). GangSim: a Simulator for Grid Scheduling Studies. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid* (pp. 1151–1158). Cardiff.

Eberhart, R., & Kennedy, J. (1995). A New Optimizer Using Particle Swarm Theory. In *Proceddings of the 6th International Symposium on Micromachine and Human Science* (pp. 39–43). Nagoya.

Espling, D. (2013). *Enabling Technologies for Management of Distributed Computing Infrastructures*. (Doctoral dissertation). Retrieved from http://www.diva-portal.org/smash/search.jsf

Farina, A., Graziano, A., Panzieri, S., Pascucci, F., & Setola, R. (2013). How to Perform Verification and Validation of Critical Infrastructure Modeling Tools. In *Proceedings of the 6th International Workshop on Critical Information Infrastructure Security* (pp. 116–127). Lucerne. doi:10.1007/978-3-642-41476-3_10

Feitelson, D. G. (2013). *Workload Modeling for Computer Systems Performance Evaluation*. Jerusalem: The Hebrew University of Jerusalem.

Fidanova, S. (2006). Simulated Annealing for Grid Scheduling Problem. In *Proceedings of the International Symposium on Modern Computing* (pp. 41–45). Sofia.

Fidanova, S., & Durchova, M. (2006). Ant Algorithm for Grid Scheduling Problem. In *Proceedings of the 5th International Conference on Large-Scale Scientific Computing* (pp. 405–412). Sozopol.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1996). *Artificial Intelligence Through Simulated Evolution*. New York: Wiley.

Foster, I., & Kesselman, C. (1997). Globus: a Metacomputing Infrastructure Toolkit. *International Journal of High Performance Computing Applications*, *11*(2), 115–128.

Foster, I., & Kesselman, C. (2004). *The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure*. Amsterdam Boston: Morgan Kaufmann.

Framinan, J. M., Leisten, R., & García, R. R. (2014). *Manufacturing Scheduling Systems An Integrated View on Models, Methods and Tools*. London: Springer.

Fulop, N. Z. C. (2008). *A Desktop Grid Computing Approach for Scientific Computing and Visualization*. (Doctoral dissertation). Retrieved from http://www.diva-portal.org/smash/search.jsf

Gainaru, A., Cappello, F., Trausan-matu, S., & Kramer, B. (2011). Event Log Mining Tool for Large Scale HPC Systems. In *Proceedings of the 17th International Euro-Par Conference on Parallel Processing* (pp. 52–64). Bordeaux, France.

Gambardella, L. M. L. M., Montemanni, R., & Weyland, D. (2012). An Enhanced Ant Colony System for the Sequential Ordering Problem. *Proceedings of the International Conference on Operations Research*, 355–360. doi:10.1007/978-3-642-

Garrido, J. M. (2001). *Object-Oriented Discrete-Event Simulation with Java: A Practical Introduction*. New York: Kluwer Academic. doi:10.1007/978-1-4615-1319-3

Gauci, M., Dodd, T. J., & Groß, R. (2012). Why "GSA: a Gravitational Search Algorithm" is not Genuinely Based on the Law of Gravity. *Journal of Natural Computing*, *11*(4), 719–720. doi:10.1007/s11047-012-9322-0

Gazi, V., & Passino, K. M. (2011). *Swarm Stability and Optimization*. Heidelberg: Springer.

Ge, H., & Tan, G. (2012). A Cooperative Intelligent Approach for Job-shop Scheduling Based on Bacterial Foraging Strategy and Particle Swarm Optimization. In C. Kahraman (Ed.), *Computational Intelligence Systems in Industrial Engineering* (pp. 363–383). Paris: Atlantis Press.

Gen, M., Zhang, W., Lin, L., & Jo, J. (2014). Recent Advances in Multiobjective Genetic Algorithms for Manufacturing Scheduling Problems. In *Proceedings of the 8th International Conference on Management Science and Engineering Management* (pp. 815–831). Lisbon.

Gendreau, M., & Potvin, J. (2014). Tabu Search. In E. K. Burke & G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 243–263). Boston, MA: Springer.

Gendreau, M., & Potvin, J.-Y. (2010). *Handbook of Metaheuristics*. New York: Springer.

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Journal of Computers and Operations Research*, *13*(5), 533 – 549. doi:10.1016/0305-0548(86)90048-1

Glover, F., & Laguna, M. (1997). *Tabu Search*. Boston: Kluwer Academic.

Golshanara, L., Rankoohi, S. M. T. R., & Shah-Hosseini, H. (2013). A Multi-Colony Ant Algorithm for Optimizing Join Queries in Distributed Database Systems. *Journal of Knowledge and Information Systems*, *39*(1), 75–206.

Grosan, C., & Abraham, A. (2007). Hybrid Evolutionary Algorithms : Methodologies , Architectures , and Reviews. In A. Abraham, C. Grosan, & H. Ishibuchi (Eds.), *Hybrid Evolutionary Algorithms* (pp. 1–17). Heidelberg: Springer.

Guo, Y., & Wang, X. (2010). Application of Simulated Annealing Algorithm toGrid Computing Scheduling based on GridSim. In *Proceedings of the 2nd International Conference on Information Science and Engineering* (pp. 1021–1024). Hangzhou, China: Ieee. doi:10.1109/ICISE.2010.5691633

Hamid, Z. A., Musirin, I., Rahim, M. N. A., & Kamari, N. A. M. (2012). Application of Electricity Tracing Theory and Hybrid Ant Colony Algorithm for Ranking Bus Priority in Power System. *International Journal of Electrical Power & Energy Systems*, *43*(1), 1427–1434.

Hao, Y., Liu, G., & Wen, N. (2012). An Enhanced Load Balancing Mechanism Based on Deadline Control on GridSim. *Journal of Future Generation Computer Systems*, *28*(4), 657–665. doi:10.1016/j.future.2011.10.010

Harchol-Balter, M., Crovella, M. E., & Murta, C. D. (1999). On Choosing a Task Assignment Policy for a Distributed Server System. *Journal of Parallel and Distributed Computing*, *59*(2), 204–228.

Hartmann, J., Makuschewitz, T., Frazzon, E. M., & Scholz-Reiter, B. (2014). A Genetic Algorithm for the Integrated Scheduling of Production and Transport Systems. In *Proceedings of of the International Annual Conference of the German Operations Research Society* (pp. 533–539). Leibniz University of Hannover, Germany.

Heien, E., Kondo, D., Gainaru, A., LaPine, D., Kramer, B., & Cappello, F. (2011). Modeling and Tolerating Heterogeneous Failures in Large Parallel Systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (pp. 1–11). Seatle, WA.

Herd, B., Miles, S., McBurney, P., & Luck, M. (2014). Verification and Validation of Agent-Based Simulations Using Approximate Model Checking. In *Proceedings of the International Workshop on Multi-Agent-Based Simulation XIV* (pp. 53–70). Saint Paul. doi:10.1007/978-3-642-54783-6 4,

Hodnefjell, S., & Junior, I. C. (2012). Classification Rule Discovery with Ant Colony Optimization Algorithm. In *Proceedings of the 13th International Conference on*

Intelligent Data Engineering and Automated Learning (pp. 678–687). Natal, Brazil.

Hogenboom, A., Frasincar, F., & Kaymak, U. (2013). Ant Colony Optimization for RDF Chain Queries for Decision Support. *Journal of Expert Systems with Applications*, *40*(5), 1555–1563.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. Cambridge, Mass: MIT Press.

Hussain, H., Malik, S. U. R., Hameedb, A., Khanb, S. U., Bickler, G., Min-Allah, N., … Rayes, A. (2013). A Survey on Resource Allocation in High Performance Distributed Computing Systems. *Journal of Parallel Computing*, *39*(11), 709–736.

Iosup, A. A., Epema, D. J. H. J., Maassen, J., & Nieuwpoort, R. van. (2007). Synthetic Grid Workloads with Ibis, Koala, and Grenchmark. In S. Gorlatch & M. Danelutto (Eds.), *Integrated Research in GRID Computing SE - 20* (pp. 271–283). Pisa, Italy: Springer US.

Iosup, A., Jan, M., Sonmez, O., & Epema, D. H. J. (2007). On the Dynamic Resource Availability in Grids. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing* (pp. 26–33). Austin, Texas.

Izakian, H., Abraham, A., & Snasel, V. (2009a). Metaheuristic Based Scheduling Meta-Tasks in Distributed Heterogeneous Computing Systems. *Journal of Sensors*, *9*(7), 5339–50.

Izakian, H., Abraham, A., & Snasel, V. (2009b). Scheduling Meta-Tasks in Distributed Heterogeneous Computing Systems: A Meta-Heuristic Particle Swarm Optimization Approach. In *Proceedings of the 9th International Conference on Hybrid Intelligent Systems* (pp. 397–402). Shenyang.

Izakian, H., Abraham, A., & Snsel, V. (2009). Performance Comparison of Six Efficient Pure Heuristics for Scheduling Meta-Tasks on Heterogeneous Distributed Environments. *Journal of Neural Network World*, *6*(09), 695–711.

Izakian, H., Ladani, B. T., Abraham, A., & Snasel, V. (2010). A Discrete Particle Swarm Optimization Approach for Grid Job Scheduling. *International Journal of Innovative Computing, Information and Control*, *6*(9), 1–15.

Izakian, H., Ladani, B. T., Zamanifar, K., & Abraham, A. (2009). A Novel Particle Swarm Optimization Approach for Grid Job Scheduling. In *Proceedings of the 3rd International Conference on Information Systems, Technology and Management* (pp. 100–109). Ghaziabad. doi:10.1007/978-3-642-00405-6_14

Javadi, B., Kondo, D., Iosup, A., & Epema, D. (2013). The Failure Trace Archive: Enabling the Comparison of Failure Measurements and Models of Distributed Systems. *Journal of Parallel and Distributed Computing*, *73*(8), 1208–1223.

Javadi, B., Kondo, D., Vincent, J., & Anderson, D. P. (2009). Mining for Statistical Models of Availability in Large-Scale Distributed Systems : An Empirical Study of SETI @ Home. In *Proceedings of the IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems* (pp. 1–10). London.

Jia, Q., & Seo, Y. (2013). An Improved Particle Swarm Optimization for the Resource-Constrained Project Scheduling Problem. *The International Journal of Advanced Manufacturing Technology*, *67*(9-12), 2627–2638.

Jourdan, L., Basseur, M., & Talbi, E.-G. (2009). Hybridizing Exact Methods and Metaheuristics: A Taxonomy. *European Journal of Operational Research*, *199*(3), 620–629. doi:10.1016/j.ejor.2007.07.035

Kant, A., Sharma, A., Agarwal, S., & Chandra, S. (2010). An ACO Approach to Job Scheduling in Grid Environment. In *Proceedings of the 1st International Conference on Swarm, Evolutionary, and Memetic Computing* (pp. 286–295). Chennai. doi:10.1007/978-3-642-17563-3_35

Karaboga, D. (2005). *An Idea Based On Honey Bee Swarm For Numerical Optimization (Technical Report No TR06)*. Turkey: Erciyes University.

Karaboga, D., & Basturk, B. (2008). On the Performance of Artificial Bee Colony (ABC) Algorithm. *Journal of Applied Soft Computing*, *8*(1), 687–697.

Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1942–1948). Perth.

Kim, J.-K., Shivle, S., Siegel, H. J., Maciejewski, A., Braun, T. D., Schneider, M., … SankarYellampalli, S. (2007). Dynamically Mapping Tasks with Priorities and Multiple Deadlines in a Heterogeneous Environment. *Journal of Parallel and Distributed Computing*, *67*(1), 154–169.

Kim, S.-S., Byeon, J.-H., Liu, H., Abraham, A., & McLoone, S. (2013). Optimal Job Scheduling in Grid Computing using Efficient Binary Artificial Bee Colony Optimization. *Journal of Soft Computing*, *17*(5), 867–882. doi:10.1007/s00500-012-0957-7

Klusacek, D., & Rudova, H. (2010). The Importance of Complete Data Sets for Job Scheduling Simulations. In *Proceedings of the 15th International Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 132–153). Atlanta. doi:10.1007/978-3-642-16505-4_8

Kokilavani, T., & Amalarethinam, D. I. G. (2013). An Ant Colony Optimization Based Load Sharing Technique for Meta Task Scheduling in Grid Computing. In *Proceedings of the 2nd International Conference on Advances in Computing and Information Technology* (pp. 395–404). Chennai. doi:10.1007/978-3-642-31552-7_41

Kolasa, T., & Krol, D. (2010). ACO-GA Approach to Paper-Reviewer Assignment Problem in CMS. In *Proceedings of the International Symposium on Agent and Multi-Agent Systems: Technologies and Applications* (pp. 360–369). Gdynia, Poland.

Kolodziej, J. (2012). *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*. New York: Springer. doi:10.1007/978-3-642-28971-2

Kołodziej, J., & Khan, S. U. (2012). Multi-Level Hierarchic Genetic-Based Scheduling of Independent Jobs in Dynamic Heterogeneous Grid Environment. *Journal of Information Sciences*, *214*(1), 1–19. doi:10.1016/j.ins.2012.05.016

Kolodziej, J., Xhafa, F., Bogdanski, M., & Bogda, M. (2010). Secure and Task Abortion Aware GA-Based Hybrid Metaheuristics for Grid Scheduling. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature* (pp. 526–535). Krakow.

Kolodziej, J., Xhafa, F., & Kolanko, L. (2009). Hierarchic Genetic Scheduler of Independent Jobs in Computational Grid Environment. In *Proceedings of the 23rd European Conference on Modelling and Simulation* (pp. 1–7). Madrid.

Kołodziej, J., Xhafa, F., & Kolodziej, J. (2011). Enhancing the Genetic-Based Scheduling in Computational Grids by a Structured Hierarchical Population. *Journal of Future Generation Computer Systems*, *27*(8), 1035–1046. doi:10.1016/j.future.2011.04.011

Kothari, V., Anuradha, J., Shah, S., & Mittal, P. (2012). A Survey on Particle Swarm Optimization in Feature Selection. In *Proceedings of the 4th International Conference on Global Trends in Information Systems and Software Applications* (pp. 192–201). Vellore.

Kousalya, K., & Balasubramanie, P. (2008). An Enhanced Ant Algorithm for Grid Scheduling Problem. *International Journal of Computer Science and Network Security*, *8*(4), 262–271.

Kousalya, K., & Balasubramanie, P. (2009). To Improve Ant Algorithm's Grid Scheduling Using Local Search. *International Journal of Intelligent Information Technology Application*, *2*(2), 71–79.

Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., & Lanza, G. (2003). *Genetic Programming IV Routine Human-Competitive Machine Intelligence*. New York: Springer.

Krakov, D., & Feitelson, D. G. (2013). High-Resolution Analysis of Parallel Job Workloads. In *Proceedings of the 16th International Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 178–195). Shanghai.

Kromer, P., Platos, J., & Snasel, V. (2012). Independent Task Scheduling by Artificial Immune Systems, Differential Evolution, and Genetic Algorithms. In

Proceedings of the 4th International Conference on Intelligent Networking and Collaborative Systems (pp. 28–32). Bucharest.

Kromer, P., Snasel, V. V., Platos, J., Abraham, A., & Izakian, H. (2009). Scheduling Independent Tasks on Heterogeneous Distributed Environments by Differential Evolution. In *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems* (pp. 170–174). Barcelona.

Ku-Mahamud, K. R., & Alobaedy, M. M. (2012). New Heuristic Function in Ant Colony System for Job Scheduling in Grid Computing. In *Proceedings of the 17th International Conference on Applied Mathematics* (pp. 47–52). Montreux.

Ku-Mahamud, K. R., & Nasir, H. J. A. (2010). Ant Colony Algorithm for Job Scheduling in Grid Computing. In *Proceedings of the 4th Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation* (pp. 40–45). Kota Kinabalu. doi:10.1109/AMS.2010.21

Kumar, S., Kumar, N., & Kumar, P. (2011). Genetic Algorithm for Network-Aware Job Scheduling in Grid Environment. In *Proceedings of the Recent Advances in Intelligent Computational Systems* (pp. 615–620). Trivandrum.

Li, H., Groep, D., Wolters, L., & Templon, J. (2006). Job Failure Analysis and Its Implications in a Large-scale Production Grid. In *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing* (pp. 1–8). Amsterdam.

Li, J., & Pan, Y. (2012). A Hybrid Discrete Particle Swarm Optimization Algorithm for Solving Fuzzy Job Shop Scheduling Problem. *The International Journal of Advanced Manufacturing Technology*, *66*(4), 583–596.

Li, X., Liao, J., & Cai, M. (2011). Ant Colony Algorithm for Large Scale TSP. In *Proceedings of the International Conference on Electrical and Control Engineering* (pp. 573–576). Yichang.

Lim, C. P., & Jain, L. C. (2009). Advances in Swarm Intelligence. In C. P. Lim, L. C. Jain, & S. Dehuri (Eds.), *Innovations in Swarm Intelligence*. Heidelberg: Springer.

Liu, D., Ma, S., Guo, Z., & Wang, X. (2012). Research of Grid Resource Scheduling Based on Improved Ant Colony Algorithm. In *Proceedings of the Third International Conference on Information Computing and Applications* (pp. 480–487). Chengde, China.

Liu, H., Abraham, A., & Hassanien, A. E. (2010). Scheduling Jobs on Computational Grids using a Fuzzy Particle Swarm Optimization Algorithm. *Journal of Future Generation Computer Systems*, *26*(8), 1336–1343. doi:10.1016/j.future.2009.05.022

Liu, J., Chen, L., Dun, Y., Liu, L., & Dong, G. (2008). The Research of Ant Colony and Genetic Algorithm in Grid Task Scheduling. In *Proceedings of the*

International Conference on MultiMedia and Information Technology (pp. 47–49). Three Gorges: Ieee.

Liu, K., Chen, J., Jin, H., & Yang, Y. (2009). A Min-Min Average Algorithm for Scheduling Transaction-Intensive Grid Workflows. In *Proceedings of the 7th Australasian Symposium on Grid Computing and e-Research* (pp. 41–48). Wellington.

Liu, L., Song, Y., & Dai, Y. (2010). Cooperative Multi-Ant colony Pseudo-Parallel Optimization Algorithm. In *Proceedings of the IEEE International Conference on Information and Automation* (pp. 1269–1274). Harbin. doi:10.1109/ICINFA.2010.5512118

Liu, X., Yi, H., & Ni, Z. (2013). Application of Ant Colony Optimization Algorithm in Process Planning Optimization. *Journal of Intelligent Manufacturing*, *24*(1), 1–13.

Liusuqin, Shuojun, Menglingfen, & Lixingsheng. (2009). "Making Concessions in Order to Gain Advantages" Improved Ant Colony Optimization for Improving Job Scheduling Problems. In *Proceedings of the Global Congress on Intelligent Systems* (pp. 115–118). Xiamen.

Lope, J. de, Maravall, D., & Quinonez, Y. (2012). Decentralized Multi-tasks Distribution in Heterogeneous Robot Teams by Means of Ant Colony Optimization and Learning Automata. In *Proceedings of the 7th International Conference on Hybrid Artificial Intelligent Systems* (pp. 103–114). Salamanca, Spain.

Lorpunmanee, S., Sap, M. N., Abdullah, A. H., & Chompoo-inwai, C. (2007). An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment. *International Journal of Computer and Information Science and Engineering*, *1*(4), 314–321.

Lublin, U., & Feitelson, D. G. (2003). The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. *Journal of Parallel and Distributed Computing*, *63*(11), 1105–1122. doi:10.1016/S0743-7315(03)00108-4

Ma, L., Lu, Y., Zhang, F., & Sun, S. (2012). Dynamic Task Scheduling in Cloud Computing Based on Greedy Strategy. In *Proceedings of the International Conference on Trustworthy Computing and Services* (pp. 156–162). Beijing.

Ma, T., Yan, Q., Liu, W., Guan, D., & Lee, S. (2011). Grid Task Scheduling: Algorithm Review. *Journal of IETE Technical Review*, *28*(2), 158–167.

MadadyarAdeh, M., & Bagherzadeh, J. (2011). An Improved Ant Algorithm for Grid Scheduling Problem Using Biased Initial Ants. In *Proceedings of the 3rd International Conference on Computer Research and Development* (pp. 373–378). Shanghai.

Magoules, F., Nguyen, T.-M.-H., & Yu, L. (2009). *Grid Resource Management : Toward Virtual and Services Compliant Grid Computing*. Boca Raton: CRC Press.

Magoules, F., Pan, I., Tan, K.-A., & Kumar, A. (2009). *Introduction to Grid Computing*. Boca Raton: CRC Press.

Maheshbhai, L. A. (2011). Job Scheduling Based on Reliability, Time and Cost Constraints under Grid Environment. In *Proceedings of the Nirma University International Conference on Engineering* (pp. 1–5). Ahmedabad.

Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., & Freund, R. F. (1999). Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Proceedings of the 8th Heterogeneous Computing Workshop* (pp. 30–44). San Juan.

Malarvizhi, N., & Uthariaraj, V. R. (2009). A Minimum Time to Release Job Scheduling Algorithm in Computational Grid Environment. In *Proceedings of the 5th International Joint Conference on INC, IMS and IDC* (pp. 13–18). Seoul.

Mao, J. (2011). A Task Scheduling Method of Grid Service using Ant Colony Optimization. In *Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce* (pp. 2752–2755). Zhengzhou, China.

Maruthanayagam, D., & UmaRani, R. (2010). Enhanced Ant Colony Algorithm for Grid Scheduling. *International Journal of Computer Technology and Applications*, *1*(1), 43–53.

Mathiyalagan, P., Suriya, S., & Sivanandam, S. N. (2010). Modified Ant Colony Algorithm for Grid Scheduling. *International Journal on Computer Science and Engineering*, *02*(02), 132–139.

Meihong, W., Wenhua, Z., Wang, M., & Zeng, W. (2010). A Comparison of Four Popular Heuristics for Task Scheduling Problem in Computational Grid. In *Proceedings of the 6th International Conference on Wireless Communications Networking and Mobile Computing* (pp. 1–4). Chengdu. doi:10.1109/WICOM.2010.5600872

Menasce, D. A., Saha, D., Porto, S. C. D. S., Almeida, V. A. F., & Tripathi, S. K. (1995). Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures. *Journal of Parallel and Distributed Computing*, *28*(1), 1–18. doi:10.1006/jpdc.1995.1085

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Heidelberg: Springer-Verlag.

Michalewicz, Z. (1999). *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag.

Michelakos, I., Mallios, N., Papageorgiou, E., & Vassilakopoulos, M. (2011). Ant Colony Optimization and Data Mining. In N. Bessis & F. Xhafa (Eds.), *Next Generation Data Technologies for Collective Computational Intelligence* (pp. 31–60). Heidelberg: Springer. doi:10.1007/978-3-642-20344-2_2

Montes, J., Sanchez, A., & Perez, M. S. (2012). Riding Out the Storm: How to Deal with the Complexity of Grid and Cloud Management. *Journal of Grid Computing*, *10*(3), 349–366. doi:10.1007/s10723-012-9225-4

Moscato, P., & Cotta, C. (2010). A Modern Introduction to Memetic Algorithms. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 141–183). New York: Springer.

Mou, L.-M. (2011). A Novel Ant Colony System with Double Pheromones for the Generalized TSP. In *Proceedings of the 7th International Conference on Natural Computation* (pp. 1923–1928). Shanghai.

Nagariya, S., Mishra, M., & Shrivastava, M. (2014). An Inherent Approach based on ACO and Tabu Search for Resource Allocation in Grid Environment. *International Journal of Computer Applications*, *87*(6), 39–45.

Nayak, S. K., Padhy, S. K., Panigrahi, S. P., Kumari, S., & Prasada, S. (2012). A Novel Algorithm for Dynamic Task Scheduling. *Journal of Future Generation Computer Systems*, *28*(5), 709–717. doi:10.1016/j.future.2011.12.001

Nithya, L. M., Nadu, T., & Shanmugam, A. (2011). Scheduling in Computational Grid with a New Hybrid Ant Colony Optimization Algorithm. *European Journal of Scientific Research*, *62*(2), 273–281.

Noghanian, S., Sabouni, A., Desell, T., & Ashtari, A. (2014). Global Optimization Differential Evolution, Genetic Algorithms, Particle Swarm, and Hybrid Methods. In S. Noghanian, A. Sabouni, T. Desell, & A. Ashtari (Eds.), *Microwave Tomography Global Optimization, Parallelization and Performance Evaluation* (pp. 39–61). New York: Springer.

Nothegger, C., Mayer, A., Chwatal, A., & Raidl, G. R. (2012). Solving the Post Enrolment Course Timetabling Problem by Ant Colony Optimization. *Journal of Annals of Operations Research*, *194*(1), 325–339. doi:10.1007/s10479-012-1078-5

Pace, D. K. (2003). Verification, Validation, and Accreditation of Simulation Models. In M. S. Obaidat & G. I. Papadimitriou (Eds.), *Applied System Simulation Methodologies and Applications*. New York: Springer.

Passos, W. Dos. (2009). *Numerical Methods, Algorithms and Tools in C#*. Boca Raton: CRC Press.

Pereira, C., Goncalves, L., & Ferreira, M. (2013). Optic Disc Detection in Color Fundus Images Using Ant Colony Optimization. *Journal of Medical & Biological Engineering & Computing*, *51*(3), 295–303.

Phatanapherom, S., Uthayopas, P., & Kachitvichyanukul, V. (2003). Fast Simulation Model for Grid Scheduling Using Hypersim. In *Proceedings of the Winter Simulation Conference* (pp. 1494 – 1500). New Orleans, LA.

Pintea, C.-M. (2014). *Advances in Bio-inspired Computing for Combinatorial Optimization Problems*. Berlin Heidelberg: Springer.

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle Swarm Optimization. *Journal of Swarm Intelligence*, *1*(a), 33–57. doi:10.1007/s11721-007-0002-0

Pooranian, Z., Shojafar, M., Abawajy, J. H., & Abraham, A. (2013). An efficient meta-heuristic algorithm for grid computing. *Journal of Combinatorial Optimization*, 1–22. doi:10.1007/s10878-013-9644-6

Price, K. V., Storn, R. M., & Lampinen, J. A. (2005). *Differential Evolution A Practical Approach to Global Optimization*. Berlin: Springer.

Qureshi, M. B., Dehnavi, M. M., Min-Allah, N., Qureshi, M. S., Hussain, H., Rentifis, I., … Zomaya, A. Y. (2014). Survey on Grid Resource Allocation Mechanisms. *Journal of Grid Computing*, *12*(2), 399–441. doi:10.1007/s10723-014-9292-9

Rajni, & Chana, I. (2013). Bacterial Foraging Based Hyper-Heuristic for Resource Scheduling in Grid Computing. *Journal of Future Generation Computer Systems*, *29*(3), 751–762. doi:10.1016/j.future.2012.09.005

Raju, R., Babukarthik, R. G., & Dhavachelvan, P. (2013). Hybrid Ant Colony Optimization and Cuckoo Search Algorithm for Job Scheduling. In *Proceedings of the 2nd International Conference on Advances in Computing and Information Technology* (pp. 491–501). Chennai.

Reeves, C. R., & Rowe, J. E. (2003). *Genetic Algorithms: Principles and Perspectives A Guide to GA Theory*. Boston: Kluwer Academic Publishers.

Ritchie, G., & Levine, J. (2003). A Fast, Effective Local Search for Scheduling Independent Jobs in Heterogeneous Computing Environments. In *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group* (pp. 178–183). Glasgow.

Ritchie, G., & Levine, J. (2004). A Hybrid Ant Algorithm for Scheduling Independent Jobs in Heterogeneous Computing Environments. In *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group* (pp. 1–7). Cork.

Rothlauf, F. (2011). *Design of Modern Heuristics Principles and Application*. Heidelberg: Springer.

Salman, A., Ahmad, I., & Al-Madani, S. (2002). Particle Swarm Optimization for Task Assignment Problem. *Journal of Microprocessors and Microsystems*, *26*(8), 363–371.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. New York: Wiley.

See, P. C., Tai, V. C., & Molinas, M. (2012). Ant Colony Optimization Applied to Control of Ocean Wave Energy Converters. *Journal of Energy Procedia*, *20*(1), 148–155.

Selvi, S., & Manimegalai, D. (2010). Scheduling Jobs on Computational Grid using Differential Evolution Algorithm. In *Proceedings of the 12th International Conference on Networking, VLSI and Signal Processing* (pp. 118–123). University of Cambridge, UK.

Seo, K.-K. (2012). An Ant Colony Optimization Algorithm Based Image Classification Method for Content-Based Image Retrieval in Cloud Computing Environment. In *Proceedings of the International Conferences on Computer Applications for Web, Human Computer Interaction, Signal and Image Processing, and Pattern Recognition* (pp. 110–117). Jeju Island.

Shang, J., Zhang, J., Lei, X., Zhang, Y., & Chen, B. (2012). Incorporating Heuristic Information into Ant Colony Optimization for Epistasis Detection. *Journal of Genes & Genomics*, *34*(3), 321–327.

Silva, D. P. da, Cirne, W., & Brasileiro, F. V. (2003). Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In *Proceedings of the 9th International Euro-Par Conference on Parallel Processing* (pp. 169–180). Klagenfurt. doi:10.1007/978-3-540-45209-6_26

Sim, K. M. (2009). Special Issue on Grid Resource Management. *IEEE SYSTEMS JOURNAL*, *3*(1), 2–5.

Singh, L., & Singh, S. (2014). A Genetic Algorithm for Scheduling Workflow Applications in Unreliable Cloud Environment. In *Proceedings of the 2nd International Conference on Recent Trends in Computer Networks and Distributed Systems Security* (pp. 139–150). Trivandrum.

Sivanandam, S. N., & Deepa, S. N. (2008). *Introduction to Genetic Algorithms*. *Electronics* (Vol. 2, p. 442). Heidelberg: Springer. doi:10.1007/978-3-540-73190-0

Smith, A. J. (2007). Workloads. *Journal of Communications of the ACM*, *50*(11), 45–50.

Song, X., Sun, L., & Cao, Y. (2010). Study on the Convergence of Converse Ant Colony Algorithm for Job Shop Scheduling Problem. In *Proceedings of the 6th International Conference on Natural Computation* (pp. 2710–2714). Yantai.

Sonmez, O., Yigitbasi, N., Abrishami, S., Iosup, A., & Epema, D. (2010). Performance Analysis of DynamicWorkflow Scheduling in Multicluster Grids. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (pp. 49–60). Chicago.

Stoean, C., & Stoean, R. (2014). *Support Vector Machines and Evolutionary Algorithms for Classification*. Switzerland: Springer.

Stutzle, T., & Hoos, H. (1997). MAX-MIN Ant System and Local Search for the Traveling Salesman Problem. In *Proceedings of the International Conference on Evolutionary Computation* (pp. 309–314). Indianapolis. doi:10.1109/ICEC.1997.592327

Stutzle, T., & Hoos, H. H. (2000). MAX-MIN Ant System. *Journal of Future Generation Computer Systems*, *16*(8), 889–914. doi:10.1016/S0167-739X(00)00043-1

Talbi, E. (2013a). A Unified Taxonomy of Hybrid Metaheuristics with Mathematical Programming, Constraint Programming and Machine Learning. In E. Talbi (Ed.), *Hybrid Metaheuristics*. Heidelberg: Springer.

Talbi, E. (2013b). *Hybrid Metaheuristics*. Heidelberg: Springer.

Thesen, A. (1998). Design and Evaluation of Tabu Search Algorithms for Multiprocessor Scheduling. *Journal of Heuristics*, *4*(2), 141–160.

Tian, J., Yu, W., Chen, L., & Ma, L. (2011). Image Edge Detection Using Variation-Adaptive Ant Colony Optimization. In N. T. Nguyen (Ed.), *Transactions on Computational Collective Intelligence V* (pp. 27–40). New York: Springer.

Tian, Y., Liu, D., Yuan, D., & Wang, K. (2012). A Discrete PSO for Two-Stage Assembly Scheduling Problem. *The International Journal of Advanced Manufacturing Technology*, *66*(4), 481–499.

Tiwari, P., & Verma, B. (2012). Application of Ant Colony Algorithm for Classification and Rule Generation of Data. In S. Patnaik & Y.-M. Yang (Eds.), *Soft Computing Techniques in Vision Science* (pp. 155–170). Berlin: Springer. doi:10.1007/978-3-642-25507-6_14

Tsutsui, S., & Fujimoto, N. (2013). ACO withTabu Searchon GPUs for Fast Solution of the QAP. In S. Tsutsui & P. Collet (Eds.), *Massively Parallel Evolutionary Computation on GPGPUs* (pp. 179–202). Heidelberg: Springer.

Ullman, J. D. (1975). NP-Complete Scheduling Problems. *Journal of Computer and System Sciences*, *10*(3), 384–393.

Url, S., Archive, T. J., Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Journal of Science*, *220*(4598), 671–680.

Vidal, R. V. V. (1993). *Applied Simulated Annealing*. Berlin Heidelberg: Springer.

Visalakshi, P., & Sivanandam, S. N. (2009). Dynamic Task Scheduling with Load Balancing using Hybrid Particle Swarm Optimization. *International Journal of Open Problems Compt Math*, *2*(3), 475–488.

Vob, S. (2001). Meta-Heuristics: The State of the Art. In *Proceedings of the Workshop on Local Search for Planning and Scheduling* (pp. 1–23). Heidelberg.

Wang, J., Duan, Q., Jiang, Y., & Zhu, X. (2010). A New Algorithm for Grid Independent Task Schedule : Genetic Simulated Annealing. In *Proceedings of the World Automation Congress* (pp. 165–171). Kobe.

Wang, Y., Zhang, J., Zhao, Y., Wang, J., & Gu, W. (2013). ACO-Based Routing and Spectrum Allocation in Flexible Bandwidth Networks. *Journal of Photonic Network Communications*, *25*(3), 135–143. doi:10.1007/s11107-013-0397-z

Wang, Z., Jing, X., & Wang, J. (2012). A Novel Routing Algorithm Based on Ant Colony Optimization for Hierarchical Wireless Sensor Networks. In *Proceedings of the International Conference on Electrics, Communication and Automatic Control* (pp. 825–831). Chongqing, China.

Wankar, R. (2008). Grid Computing with Globus: an Overview and Research Challenges. *International Journal of Computer Science and Applications*, *5*(3), 56–69.

Wei, L., Zhang, X., Li, Y., & Li, Y. (2012). An Improved Ant Algorithm for Grid Task Scheduling Strategy. *Journal of Physics Procedia*, *24*(1), 1974–1981. doi:10.1016/j.phpro.2012.02.290

Wiener, R., & Of, O. (2009). Ant Colony System Optimization. *Journal of Object Technology*, *8*(6), 39–58.

Wolski, R., Spring, N. T., & Hayes, J. (1999). The Network Weather Service: a Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computer Systems*, *15*(5-6), 757–768. doi:10.1016/S0167-739X(99)00025-4

Wu, C., Zhang, N., Jiang, J., Yang, J., & Liang, Y. (2007). Improved Bacterial Foraging Algorithms and Their Applications to Job Shop Scheduling Problems. In *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms* (pp. 562–569). Warsaw.

Xhafa, F. (2006). An Experimental Study on GA Replacement Operators for Scheduling on Grids. In *Proceedings of the 2nd International Conference on Bioinspired Optimization Methods and their Applications* (pp. 121–130). Ljubljana.

Xhafa, F. (2007). A Hybrid Evolutionary Heuristic for Job Scheduling on Computational Grids. In A. Abraham, C. Grosan, & H. Ishibuchi (Eds.), *Hybrid Evolutionary Algorithms* (pp. 269–311). Heidelberg: Springer.

Xhafa, F., & Abraham, A. (2008a). Meta-heuristics for Grid Scheduling Problems. In F. Xhafa & A. Abraham (Eds.), *Metaheuristics for Scheduling in Distributed Computing Environments* (pp. 1–37). Heidelberg: Springer. doi:10.1007/978-3-540-69277-5_1

Xhafa, F., & Abraham, A. (2008b). *Metaheuristics for Scheduling in Distributed Computing Environments*. Berlin: Springer.

Xhafa, F., & Abraham, A. (2009). A Compendium of Heuristic Methods for Scheduling in Computational Grids. In *Proceedings of the 10th International Conference on Intelligent Data Engineering and Automated Learning* (pp. 751–758). Burgos. doi:10.1007/978-3-642-04394-9_92

Xhafa, F., & Abraham, A. (2010). Computational Models and Heuristic Methods for Grid Scheduling Problems. *Journal of Future Generation Computer Systems*, *26*(4), 608–621. doi:doi:10.1016/j.future.2009.11.005

Xhafa, F., Alba, E., & Dorronsoro, B. (2007). Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium* (pp. 1 – 8). Long Beach, CA.

Xhafa, F., Alba, E., Dorronsoro, B., Duran, B., & Abraham, A. (2008). Efficient Batch Job Scheduling in Grids Using Cellular Memetic Algorithms. *Journal of Mathematical Modelling and Algorithms*, *7*(2), 217–236.

Xhafa, F., Barolli, L., & Durresi, A. (2007a). An Experimental Study on Genetic Algorithms for Resource Allocation on Grid Systems. *Journal of Interconnection Networks*, *8*(4), 427–443.

Xhafa, F., Barolli, L., & Durresi, A. (2007b). Batch Mode Scheduling in Grid Systems. *International Journal of Web and Grid Services*, *3*(1), 19–37.

Xhafa, F., Barolli, L., & Durresi, A. (2007c). Immediate Mode Scheduling of Independent Jobs in Computational Grids. In *Proceedings of the 21st International Conference on Advanced Networking and Applications* (pp. 970–977). Niagara Falls.

Xhafa, F., & Carretero, J. (2009). Experimental Study of GA-Based Schedulers in Dynamic Distributed Computing Environments. In E. Alba, C. Blum, P. Isasi, C. Leon, & J. A. Gomez (Eds.), *Optimization Techniques for Solving Complex Problems* (pp. 423–441). Hoboken, N.J: Wiley.

Xhafa, F., Carretero, J., Alba, E., & Dorronsoro, B. (2008). Design and Evaluation of Tabu Search Method for Job Scheduling in Distributed Environments. In *Proceedings of the International Symposium on Parallel and Distributed Processing* (pp. 1 – 8). Miami, FL.

Xhafa, F., Carretero, J., Barolli, L., & Durresi, A. (2007). Requirements for an Event-Based Simulation Package for Grid Systems. *Journal of Interconnection Networks*, *08*(02), 163–178. doi:10.1142/S0219265907001965

Xhafa, F., Carretero, J., Dorronsoro, B. B., & Alba, E. (2009). A Tabu Search Algorithm for Scheduling Independent Jobs in Computational Grids. *Journal of Computing and Informatics*, *28*(2), 237–250.

Xhafa, F., & Duran, B. (2008). Parallel Memetic Algorithms for Independent Job Scheduling in Computational Grids. In C. Cotta & J. van Hemert (Eds.), *Recent Advances in Evolutionary Computation for Combinatorial Optimization* (pp. 219–239). Heidelberg: Springer. doi:10.1007/978-3-540-70807-0_14

Xhafa, F., Duran, B., Abraham, A., & Dahal, K. P. (2008). Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids. In *Proceedings of the 7th Computer Information Systems and Industrial Management Applications* (pp. 275–280). Ostrava.

Xhafa, F., Duran, B., & Kolodziej, J. (2011). On Exploitation vs Exploration of Solution Space for Grid Scheduling. In *Proceedings of the 3rd International Conference on Intelligent Networking and Collaborative Systems* (pp. 164–171). Fukuoka. doi:10.1109/INCoS.2011.128

Xhafa, F., Gonzalez, J. A., Dahal, K. P., & Abraham, A. (2009). A GA(TS) Hybrid Algorithm for Scheduling in Computational Grids. In *Proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems* (pp. 285–292). Salamanca. doi:10.1007/978-3-642-02319-4_34

Xhafa, F., Kolodziej, J., Barolli, L., & Fundo, A. (2011). A GA+TS Hybrid Algorithm for Independent Batch Scheduling in Computational Grids. In *Proceedings of the 14th International Conference on Network-Based Information Systems* (pp. 229–235). Tirana. doi:10.1109/NBiS.2011.41

Xhafa, F., Kolodziej, J., Barolli, L., Kolici, V., Miho, R., & Takizawa, M. (2011). Evaluation of Hybridization of GA and TS Algorithms for Independent Batch Scheduling in Computational Grids. In *Proceedings of the International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (pp. 148–155). Barcelona. doi:10.1109/3PGCIC.2011.31

Xhafa, F., Koodziej, J., Duran, B., Bogdanski, M., & Barolli, L. (2011). A Comparison Study on the Performance of Population-based Meta-Heuristics for Independent Batch Scheduling in Grid Systems. In *Proceedings of the International Conference on Complex Intelligent and Software Intensive Systems* (pp. 123–130). Seoul.

Xing, B., & Gao, W.-J. (2014). Bacteria Inspired Algorithms. In B. Xing & W.-J. Gao (Eds.), *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms* (pp. 21–38). Cham: Springer.

Yan, H. U. I., Shen, X., Li, X., & Wu, M. (2005). An Improved Ant Algorithm for Job Scheduling in Grid Computing. In *Proceedings of the 4th International Conference on Machine Learning and Cybernetics* (pp. 2957–2961). Guangzhou. doi:10.1109/ICMLC.2005.1527448

Yang, X.-S. (2014). *Nature-Inspired Optimization Algorithms*. Amsterdam: Elsevier.

YarKhan, A., & Dongarra, J. J. (2002). Experiments with Scheduling Using Simulated Annealing in a Grid Environment. In *Proceedings of the 3rd International Workshop on Grid Computing* (pp. 232–242). Baltimore.

Yin, J., & Xiang, W. (2012). Ant Colony Algorithm for Surgery Scheduling Problem. In *Proceedings of the Third International Conference on Advances in Swarm Intelligence* (pp. 198–205). Shenzhen, China.

Youhui, L., Xinhua, L., & Qi, L. (2012). Assembly Sequence Planning Based on Ant Colony Algorithm. In Y. Zhang (Ed.), *Future Communication, Computing, Control and Management* (Vol. 141, pp. 397–404). Heidelberg: Springer.

Yu, H., Ni, J., & Zhao, J. (2013). ACO Sampling: An Ant Colony Optimization-Based Under sampling Method for Classifying Imbalanced DNA Microarray Data. *Journal of Neurocomputing*, *101*(1), 309–318. doi:10.1016/j.neucom.2012.08.018

Yu, J., & Wang, C. (2013). A Max–Min Ant Colony System for Assembly Sequence Planning. *International Journal of Advanced Manufacturing Technology*, *67*(9), 2819–2835.

Yu, X., & Gen, M. (2010). *Introduction to Evolutionary Algorithms*. London: Springer.

Zapfel, G., Braune, R., & Bogl, M. (2010). *Metaheuristic Search Concepts a Tutorial with Applications to Production and Logistics*. Heidelberg: Springer.

Zhang, S., Ning, T., & Zhang, Z. (2012). A New Hybrid Ant Colony Algorithm for Solving Vehicle Scheduling Problem. *International Journal of Advancements in Computing Technology*, *4*(5), 17–23.

Zhang, T., Lin, J., Qiu, B., & Fu, Y. (2011). Solving the Aircraft Assigning Problem by the Ant Colony Algorithm. In *Proceedings of the International Conference on Information and Management Engineering* (pp. 179–187). Wuhan, China.

Zheng, Q., Li, M., Li, Y., & Tang, Q. (2013). Station Ant Colony Optimization for the Type 2 Assembly Line Balancing Problem. *The International Journal of Advanced Manufacturing Technology*, *66*(9), 1859–1870.

Zhong, L., Long, Z., Zhang, J., & Song, H. (2011). An Efficient Memetic Algorithm for Job Scheduling in Computing Grid. In *Proceedings of the International Symposium on Information and Automation* (pp. 650–656). Guangzhou.

Zhu, P., Zhao, M., & He, T. (2010a). A Novel Ant Colony Algorithm for Grid Task Scheduling. *Journal of Computational Information Systems*, *6*(3), 745–752.

Zhu, P., Zhao, M., & He, T. (2010b). A Novel Ant Colony Optimization Algorithm in Application of Pheromone Diffusion. In *Proceedings of the International Conference on Life System Modeling and Simulation* (pp. 1–8). Wuxi, China.

Zhu, Y., & Wei, Q. (2010). An Improved Ant Colony Algorithm for Independent Tasks Scheduling of Grid. In *Proceedings of the 2nd International Conference on Computer and Automation Engineering* (pp. 566–569). Singapore.