

UNIVERSITY OF BAHRAIN

College of Information Technology



Energy-Aware Scheduling in Multiprocessor Real-Time Systems

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Master Degree in Information Technology

Submitted by

Noor Adnan Awwad

University Number: 20027564

Supervised by

Dr. Amjad Mahmood

(Associate Professor)

University of Bahrain

Co-Supervisor

Dr. Salman Khan

(Assistant Professor)

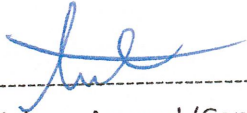
University of Bahrain

Kingdom of Bahrain

October, 2016

DECLARATION

I declare that this work is the result of my own investigation and that it has not already been accepted in substance for any degree, nor is it currently submitted for any degree.

Signed: 
Noor Adnan Awwad (Candidate)

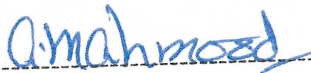
Date: 23 Nov 2016

The Thesis defense committee considers the Thesis titled:

Energy-Aware Scheduling in Multiprocessor Real-Time Systems

Submitted by Noor Adnan Awwad to the College of Information Technology is satisfactory and acceptable for the Master of Science in Information Technology.

Thesis Defense Committee:


Dr. Amjad Mahmood
(Supervisor)

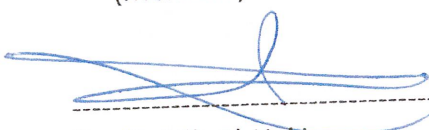
Department of Computer Science
University of Bahrain


Dr. Salman Khan
(Co-Supervisor)

Department of Computer Science
University of Bahrain


Prof. Jamil Ahmad
(Member)

Abasyn University
Pakistan


Dr. Riyadh Al-Hakim
(Member)

Department of Computer Science
University of Bahrain

Abstract

Energy consumption is becoming one of the most critical issues in various system designs. In multiprocessor systems, the energy consumption is highly affected by task allocation approaches. In the last decades designing an efficient system with lower power consumption has captured the attention of the research community in the area of computer science. The problem is considered as a design goal in limited power systems and mobile systems, particularly for real-time systems.

Power-aware scheduling of real-time tasks using dynamic scaling on multiprocessor systems is addressed in this thesis. Real-time task allocation problem is presented as an optimization problem subjected to a number of constraints. Many algorithms have been developed to solve the optimization problem in order to achieve optimal or semi optimal results. Nature-inspired algorithms such as Genetic algorithm (GA) and Cuckoo search (CS) are adapted in this research to solve the underlying problem. Parameter sensitivity analysis is used to decide the most appropriate parameters' value and combinations for both algorithms. Several variations of the crossover and mutation operators in GA are proposed. For the CS algorithm, an approach for determining suitable values of its parameters is suggested. The research emphasizes the effect of different parameters and operators on algorithms performance. The efficiency of algorithms is compared in terms of solution quality and execution time. The most appropriate parameters' value and combinations are then determined for each algorithm. A variation of basic CS (called CS with penalty) is also proposed in which infeasible solutions are allowed to be part of the population with a penalty. The adapted approaches provide savings up to 89% for the best cases in power consumption. GA and CS in most cases produced similar results in terms of solution quality. However GA slightly outperforms CS in terms of execution time. In general, simulation results on multiprocessor system composed of 5 processors show that GA performs better than CS by a percentage up to 58% also GA is performing better than CS with penalty by percentage up to 50% in almost all cases.

To my beloved husband and supportive family

Acknowledgment

First and foremost, praises and thanks to Allah for his guidance all through the stages of my research and given me the strength, patience, knowledge and ability to complete my work successfully.

I would like to express my sincere gratitude to my advisors Dr.Amjad Mahmood and Dr.Salman Khan for the continuous support of my master study and related research, for their patience, motivation, and immense knowledge. Their guidance helped throughout my research and writing of this thesis.

Finally, I would like to thank my supportive parents, sister and husband for their continuous support, encouragement and limitless patience to help me throughout writing this thesis and my life in general.

Table of Contents

Subject	Page
Defense Committee's Approval	I
Abstract	II
Dedication	III
Acknowledgment	IV
Table of Contents	V
List of Tables	VIII
List of Figures	IX
List of Symbols	XII
Chapter One Introduction	
1.1 Background	2
1.2 Real-Time Systems	3
1.3 Task Allocation and Scheduling	6
1.3.1 Scheduling Tasks on Multiprocessor System	6
1.4 Dynamic Voltage Scaling	7
1.5 Problem Statement	8
1.6 Motivating Example	9
1.7 Research Objectives	12
1.8 Methodology	13
1.9 Research Contributions	14
1.10 Thesis Outline	15
Chapter Two Literature Review	
2.1 Task Allocation in Distributed and Grid Systems	17
2.2 Task Allocation and Scheduling in Multiprocessor Systems	21
2.2.1 General Task Allocation Approaches	22
2.2.2 Real Time Task Allocation to Minimize Energy Consumption	23
2.3 A Review of Genetic Algorithm	35
2.4 A Review of Cuckoo Search	40
2.5 Previous Work Limitations and Gap Analysis	43

2.6 Summary	45
Chapter Three Problem Formulation	
3.1 Notations	47
3.2 System Models	48
3.2.1 Task Model	48
3.2.2 Power Model	51
3.3 Objective Function and Constraints	55
3.4 Summary	56
Chapter Four Proposed Algorithms	
4.1 Genetic Algorithm	58
4.1.1 Solution Encoding	59
4.1.2 Generation of Initial Population	60
4.1.3 Crossover Operator	60
4.1.4 Mutation Operator	63
4.1.5 Replacement	66
4.1.6 Solution Validation	67
4.2 Cuckoo Search	68
4.2.1 Solution Encoding and Initial Solution Generation	70
4.2.2 Generating a Cuckoo Egg by Lévy Flight	70
4.2.3 Replace Worst Nests	72
4.2.4 Solution Validation	73
4.2.5 Dealing With Infeasible Solutions	73
4.3 Summary	73
Chapter Five Results and Discussion	
5.1 Characteristics of Test Cases	75
5.2 Experimental procedure	76
5.3 Results and discussion of Genetic Algorithm	76
5.3.1 Parameter Settings	77
5.3.2 Effect of Parameter Values	79
5.3.2.1 Effect of Population Size	79
5.3.2.2 Effects of Crossover and Mutation Operators	82
5.3.2.3 Effect of Replacement Operator	84

5.3.3 Experimentation with Selected Parameter Combinations	85
5.4 Results and Discussion of Cuckoo Search	87
5.4.1 Parameter Settings	88
5.4.2 Effect of Parameter Values	88
5.4.2.1 Effect of Number of Nests	89
5.4.2.2 Effect of λ and p_a Values	91
5.4.3 Experimentation with Selected Parameter Combinations	92
5.5 Results and Discussion of Cuckoo Search with Penalty	94
5.5.1 Parameter Setting	95
5.5.2 Effect of parameter Values	95
5.5.2.1 Effect of Number of Nests	95
5.5.2.2 Effect of λ and p_a Values	97
5.5.3 Experimentation with Selected Parameter Combinations	98
5.6 Comparison of Algorithms	100
5.6.1 Solution Quality Analysis	100
5.6.2 Comparison of Run Time	102
5.6.3 Percentage Saving in Power	104
5.6.4 Statistical Test Results	106
5.7 Effect of Changing the Number of Processors on Algorithm Performance	107
5.8 Summary	109
Chapter Six	
Conclusion and Future Work	
6.1 Conclusion	111
6.2 Future Work	113
References	115

List of Tables

No.	Subject	Page
3.1	Notations	47
5.1	Characteristics of Test cases used in experiments	76
5.2	Parameter settings for GA	77
5.3	GA combination testing set	77
5.4	Average fitness of best combinations for Test case2	84
5.5	Average execution time of best combinations for Test case 2	85
5.6	Solution quality of executing the Algorithm on different Multi-processor Systems	87
5.7	Parameter settings used in CS	88
5.8	Parameter combinations used in CS	88
5.9	Cuckoo search results for Test case 1(# of tasks 10)	90
5.10	Cuckoo search results for Test Case 2 (# of tasks 15)	90
5.11	Solution quality analysis of best parameter combinations	93
5.12	Time to reach best solution quality for selected combinations (in minutes)	94
5.13	Cuckoo search with penalty results Test case 1 (# of tasks 10)	96
5.14	Cuckoo search with penalty results Test case 2 (# of tasks 15)	96
5.15	Solution quality analysis for Test casel with best selected parameter combinations	99
5.16	Time to achieve best results analysis	100
5.17	Solution quality comparison	101
5.18	Performance analysis of average execution time in minutes	103
5.19	Time required by each algorithm to produce the best solution quality (in minutes)	104
5.20	Comparison of percentage saving using 5 processors	105
5.21	P-level value of wilcoxon statistical test	107

List of Figures

No.	Subject	Page
1.1	Schedules generated according to different scheduling methods	11
1.2	Study methodology	14
2.1	Cuckoo search algorithm	42
3.1	A task graph with 10 tasks	49
3.2	Power model example	54
4.1	Pseudo code of the adapted GA	58
4.2	Solution encoding	59
4.3	A directed acyclic graph with four tasks	59
4.4	A Chromosome	60
4.5	Generation of the initial population	60
4.6	Type 1 crossover example	61
4.7	Type 2 crossover example	62
4.8	Type 3 crossover example	63
4.9	Type 1 mutation example	64
4.10	Type 2 mutation example	64
4.11	Type 3 mutation example	65
4.12	Type 4 mutation example	66
4.13	Constraints validation algorithm	68
4.14	Proposed cuckoo search Algorithm	69
4.15	Calculate step size algorithm	70
4.16	Generate new solution algorithm	71
4.17	Example of generating new solution and stepsize calculation	72
5.1	Fitness values obtained during the 30 runs for 20, 40, and 60 chromosomes in the population for Test case 1 for the combination :Type 3 crossover (rate= 0.8), Type 1 mutation (rate= 0.1) and Type 2 replacement	80
5.2	Fitness function values obtained during the 30 runs for 20, 40, and 60 chromosomes in the population for Test case1 for the combination: Type 3 crossover (rate= 0.6), Type 3 mutation (rate= 0.05) and Type 2 replacement	81
5.3	Fitness function values obtained during the 30 runs for	81

	20, 40, and 60 chromosomes in the population for Test case2 for the combination: Type 3 crossover (rate=0.8), Type 1 mutation (rate= 0.05) and Type 1 replacement	
5.4	Fitness function values obtained during the 30 runs for 20, 40, and 60 chromosomes in the population for Test case2 for the combination: Type 2 crossover (rate=0.8), Type 1 mutation (rate= 0.05) and Type 1 replacement	82
5.5	Effect of population size on average execution time on Test case1	82
5.6	Effect of population size on average execution time on Test case2	83
5.7	Effect of GA combinations on solution quality of Test case 1 (300 iteration, 20, 40, and 60 chromosomes)	83
5.8	Effect of GA combinations on solution quality of Test case 2 (300 iteration, 20, 40, and 60 chromosomes)	83
5.9	Time analysis of all test cases with the best combinations on different multiprocessor systems	86
5.10	CS algorithm performance in terms of solution quality for Test case 2	89
5.11	Fitness function values obtained during 30 runs for 20, 40, and 60 nests (Results for Test case 2 on the combination Cc2 on 5 processors system)	91
5.12	CS performance in terms of average execution time	92
5.13	Effect of number of nests on algorithm performance in terms of solution quality on Test case 2	95
5.14	Effect of number of nests on CS with penalty algorithm performance in terms of Solution quality for Test case 2 with combination Cc2 on 5processors	97
5.15	Effect of parameter combinations on the performance in terms of Solution quality on Test case 2	97
5.16	Effect of parameter combinations on the performance in terms of Average execution time on Test case 2	98
5.17	Best combinations of GA, CS and CS with penalty comparison time analysis (in minutes)	102
5.18	Percentage power saving	105

5.19	Percentage saving comparison (using 5 processors)	106
5.20	GA combination C10 performance on different multiprocessor systems (composed of 5, 7, and 9 processors)	108
5.21	CS combination Cc4 performance on different multiprocessor systems (composed of 5, 7, and 9 processors)	108
5.22	CS with penalty combination Cc10 performance on different multiprocessor systems (composed of 5, 7, and 9 processors)	108

List of Symbols

m	number of processors
n	number of tasks
v_i	Supply voltage provided to execute task t_i
v_t	Threshold voltage
e_i	Energy consumed for processing task t_i
c_{eff}	Effective switching capacitance, where it is considered to be constant for all tasks
l_k	Number of discrete voltage levels on which processor p_j can operate on
f_i	Clock frequency at run time (operational frequency)
T	The set of all tasks; $T = \{t_1, t_2, \dots, t_n\}$.
t_i	A Task with identification number i
P	The set of Processors; $P = \{p_1, p_2, \dots, p_m\}$
p_j	Processor on which a task is mapped.
τ_i	Execution time of task t_i
c_i	Number of clock cycles required to process task t_i .
pow_i	Power consumption of executing task t_i
e_{ijk}	energy consumed of processing task t_i on processor p_j at voltage v_k and frequency f_i
E	Total energy consumed to process all the n tasks
x_{ijk}	$X = \{x_{ijk} \in \{1,0\} i \in [1, n], j \in [1, m], k \in [1, l_k]\}$ Any element of X : $x_{ijk}=1$ if a task t_i is allocated to processor p_j operating on voltage level v_k , otherwise $x_{ijk}=0$.
EST_i	Earliest start time for task t_i
LST_i	Latest start time of task t_i
FT_i	Finish time of task t_i
$Pre(t_i)$	Set of predecessors of task t_i (precedence tasks)
$Succ(t_i)$	Set of immediate successors of task t_i
ST_i	Actual start time of task i
d_i	Deadline of task t_i
DPM	Dynamic Power Management
DVFS	Dynamic Voltage/Frequency Scaling
DVS	Dynamic Voltage Scaling
EDF	earliest deadline first

<i>TGFF</i>	Task Graphs For Free
<i>DAG</i>	Directed Acyclic Graph
<i>QoS</i>	Quality of Service
<i>ACO</i>	Ant Colony Algorithm
<i>AET</i>	Actual execution time
<i>ET</i>	Estimated execution time
<i>SA</i>	Simulated Annealing
<i>TS</i>	Tabu Search
<i>RM</i>	rate monotonic
<i>MILP</i>	mixed-integer linear programming
<i>GA</i>	Genetic Algorithm
<i>CS</i>	Cuckoo Search
<i>MCS</i>	Modified Cuckoo Search
<i>BCS</i>	Binary Cuckoo Search
<i>ICS</i>	Improved Cuckoo Search
<i>WCEC</i>	worst-case execution cycles
<i>PBFTS</i>	Primary Backup Fault Tolerant Scheduling
<i>GFTS</i>	Genetic Algorithm Based Fault Tolerant Scheduler
	Set of voltage levels of processors in the multiprocessor system.
<i>VP_{pm}</i>	$VP_{p_m} = \{ VP_{p_1}, VP_{p_2}, \dots, VP_{p_m} \}$ Where m is number of processors
	Set of voltage levels that a processor p_j operates on.
<i>VP_{pj}</i>	$VP_{p_j} = \{ VL_1, VL_2, \dots, VL_{l_k} \}$ Where l_k is number of voltage levels a processor p_j operates on.
	Each level represents the voltage and frequency pair of the processor p_j at a specific voltage level VL_k .
<i>VL_k</i>	$VL_k = (v_j, f_j)$; where v_j is the voltage supply of processor p_j at voltage level VL_k and f_j is the frequency of the processor p_j at voltage level VL_k . $1 \leq k \leq l_k$

Chapter One

Introduction

1.1 Background	2
1.2 Real-Time Systems	3
1.4 Task Allocation and Scheduling	6
1.3.1 Scheduling Tasks on Multiprocessor System	6
1.4 Dynamic Voltage Scaling	7
1.5 Problem Statement	8
1.6 Motivating Example	9
1.7 Research Objectives	12
1.8 Methodology	13
1.9 Research Contributions	14
1.10 Thesis Outline	15

Chapter One

Introduction

This chapter gives a background on the task scheduling problem in multiprocessor system. It briefly presents power minimization techniques, specifically dynamic voltage scaling. It formally defines the task allocation problem to minimize the power consumption on multiprocessor systems and states the motivations of our research work. The contributions of the thesis are also highlighted. Finally, the chapter outlines the work presented in the thesis.

1.1 Background

Energy consumption is a critical issue in parallel and distributed computing platforms, and in particular an important design objective for many modern embedded systems (Kang & Ranka, 2010; Hua, Qu, & Bhattacharyya, 2003). It becomes essential and important design objective for battery-operated devices such as personal digital assistants, digital camcorders and mobile phones (Lui & Tsui, 2006; Jejurikar, 2005; Swaminathan & Chakrabarty, 2001; Shin & Kim, 2003; Zhu et al., 2003). Thus, to design an efficient system with lower power consumption is important in scenarios where power supply is a critical issue. (Lui & Tsui, 2006; Swaminathan & Chakrabarty, 2001; Shin & Kim, 2003; Zhu et al., 2003).

Reducing power consumption or managing energy saving in a system can be achieved in various ways. Many hardware techniques such as dynamic power management (DPM) and dynamic voltage/frequency

scaling (DVFS) have been proposed in which power saving is achieved through either shutting down the unused modules or slowing them down (Zhu & Meuller, 2004; Jejurikar & Gupta, 2006).

DVFS allows adjusting processor voltage and frequency at runtime. Usually, higher processor voltage and frequency lead to higher system throughput while energy reduction can be obtained using lower voltage and frequency (Alam & Kumar, 2013). The slowing down approach using frequency and voltage scaling which is based on performance requirements while meeting requirements is known to be more effective in reducing the power consumption (Kang & Ranka, 2010; Zhu & Meuller, 2004; Zhu et al., 2003; Jejurikar, 2005).

Dynamic voltage scaling (DVS) and modulation scaling techniques provide the capability to find a balance between energy saving and the computation and communications slowdowns without violating systems constraints such as timing constraints (Jejurikar, 2005).

1.2 Real-Time Systems

Real-time systems are those systems in which it should perform correctly with satisfying time constraints (Buttazzo, 2011; Davis, 2014). Real-time systems are widespread all over without being noticed. Real time systems range from very complicated and sophisticated systems to simple ones. Some examples of real-time systems include process control systems, flight control systems, flexible manufacturing applications, robotics, intelligent highway systems, and high speed and multimedia communication systems (Mohammadi & Akl, 2005; Mohammadi, 2009).

The correctness of real-time systems depends not only on logical correctness of output but also the time at which the operation completes (Davis, 2014; Alam & Kumar, 2013; Mohammadi & Akl, 2005). The system is said to be efficient if it completes on its deadline with minimum energy (Alam & Kumar, 2013). Real-time systems are not considered on-line or interactive systems since a system interacting with a human and waiting for a person's response is not real-time (Mohammadi, 2009).

Every task running on a real-time system has a deadline. Real-time systems can be classified as *soft real-time*, *hard-real time* or *firm real-time* systems (Alam & Kumar, 2013). Missing deadlines is much more critical in hard real-time systems which cannot allow missing any deadline without catastrophic consequences. The systems which are executing critical operations in specific applications like air traffic control, nuclear systems, some medical applications such as pacemakers, and large number of defense applications come under this category (Harkut & Agrawal, 2014). While soft real-time systems may allow missing some deadlines and still work correctly. These types of tasks might lead to pay penalties if the deadlines are violated but it will not result in disastrous consequences (Srinivasan et al., 1998; Mohammadi, 2009).

The applications which do not belong to either hard or soft real-time categories are called firm real-time systems. A firm deadline of this type of real-time systems is when the result is produced after the deadline is missed which is of no value, but occasional deadline misses are tolerable. This type of real-time systems is of use in applications which are serving to process some important operations such as preemptive scheduling (Harkut & Agrawal, 2014; Srinivasan et al., 1998).

Hard and soft real-time systems are formally represented as follows (Mohammadi, 2009): Consider a system consisting of a set of tasks $T = \{t_1, t_2, \dots, t_n\}$ where the finishing time of each task $t_i \in T$ is F_i . The system is said to be real-time system if there exists at least one task in the set $t_i \in T$ which falls into one of the following categories:

1. Task t_i is a *hard real-time task* if the execution of the task t_i should be completed by a given deadline d_i , i.e. $F_i \leq d_i$.
2. Task t_i is a *soft real-time task* if the later the task t_i finishes its computation after a given deadline d_i , the more penalties it pays.

Real-time systems are also classified depending on activation time of tasks as follows (Buttazzo, 2011; Halang & Stoyenko, 1994; Isovici & Fohler, 2000):

- **Periodic:** The tasks are activated (released) regularly at fixed rates (periods). A majority of sensory processing are periodic in nature.
- **Aperiodic:** The tasks are activated irregularly at some unknown time.
- **Sporadic tasks:** The tasks are activated irregularly with some known bounded rate. The bounded rate is characterized by a minimum inter-arrival period; that is, a minimum interval of time between two successive activations.

The concept of dependency between tasks in a real-time system can be described as if a task in a given real-time system requires receiving the information from another task of the system to start execution. Such a task is said to be a dependent task, otherwise it is independent. Therefore, if a task is categorized as a dependent task then the execution

of this task should be started after finishing the execution of the tasks it depends on (Mohammadi, 2009).

1.3 Task Allocation and Scheduling

Scheduling is the efficient assignment of a processor in a system to a specific task under certain constraints such as tasks deadline (Devi et al., 2014; Mohammadi, 2009). In classical scheduling theory it is also assumed that each task is to be processed by at most one processor at a time and each processor is capable of processing at most one task at a time (Mohammadi, 2009). The allocation of the system resources needs to be planned so that all demands are met by the time of their respective deadlines. This is usually done using a scheduler which implements a scheduling policy that determines how the resources of the system are allocated to the demands (Alam et al., 2016; Mohammadi, 2009).

Task scheduling can be classified into two broad categories: preemptive scheduling and non-preemptive scheduling (Jejurikar et al., 2006). In preemptive scheduling, the currently running task is preempted on the arrival of a higher priority task whereas in non-preemptive scheduling, a new task is scheduled only on the completion of the current task execution. In contrast, the execution of a non-preemptive task should be completed without interruption once it is started (Mohammadi, 2009).

1.3.1 Scheduling Tasks on Multiprocessor System

Executing various tasks simultaneously on different processors is known as parallel processing. Parallel processing is efficient computing approach for solving complex and computation intensive applications. Multiprocessor systems are classified into two types depending upon the

nature of processors as homogeneous and heterogeneous (Bokhari et al., 2016; Alam et al., 2016). Heterogeneous system includes processors with dissimilar computational speed (Alam et al., 2016).

Scheduling tasks on multiprocessor systems can be achieved by three basic techniques: partitioning, global scheduling algorithms and hybrid partition/global scheduling (Mohammadi, 2009). Task scheduled using task partitioning is different than global scheduling in the sense that it is solving the problem in a way of transforming the problem to many uni-processor scheduling problems. On the other hand, global scheduling proceeds by storing the arrived tasks in a queue which is accessed by all processors. Hence partitioning will usually produce low scheduling overhead compared to global scheduling. The third type which is a combination of partitioning and global scheduling techniques is the hybrid partition/global scheduling which mixes the techniques applied in both categories to overcome some flaws in either technique (Mohammadi, 2009).

1.4 Dynamic Voltage Scaling

Dynamic voltage scaling (DVS) is a mechanism used to reduce the energy consumed by adjusting the processing speed as a result of changing the voltage level. Modern processors are designed with voltage/frequency switching capabilities making it possible to control the DVS in order to minimize power consumption without being observed by the user. The incurred overhead can be ignored if such switching does not happen very frequently. Intel Strong ARM SA1100, and high-end ARM embedded microprocessor which are used in several devices, like Nokia N95, HTC TN II, Qualcomm MSM7200, Sony Ericsson Wseries,

Nintendo DS, iPod and Intel SpeedStep, AMD PowerNOW, Transmeta Crusoe, and the Mobile AMD are some examples of processors with voltage/frequency scaling capabilities (Chen et al., 2008; Lee et al., 1997).

In real-time systems where tasks can take tens of thousands of cycles to execute, applying voltage scaling carefully can achieve a large amount of energy saving. The objective of VS is to minimize the sum of each task's energy consumption to prolong battery life (Zhang et al., 2002).

DVS is known to be the most efficient technique used for the purpose of managing power consumption through dynamically changing the supply voltage because of its capability to save power quadratically with a linear slowdown in execution time (Goraczko et al., 2008; Lin & Ng, 2005). Thus, many researchers extensively studied the energy aware problem of scheduling real-time tasks using DVS technique (Xian, Lu & Li, 2007; Jejurikar et al., 2006).

1.5 Problem Statement

The problem considered in this research is formally defined as:

“Given a set of real-time tasks represented as directed acyclic graph (DAG) to be executed on dynamic voltage scaling multiprocessor system, determine the processor and the voltage level on which each task is executed such that the total energy consumed is minimum, subject to a number of system constraints such as time and task dependency constraints.”

The system model, task model, objective function and the constraints are discussed in Chapter 3.

1.6 Motivating Example

This section discusses our motivation for using DVS systems through an example adapted from Tavares, Maciel, Silva & Oliveira (2008). The example emphasizes on showing the difference in scheduling a set of tasks according to different scheduling methods and saving in total power consumption.

Assume a set of tasks $T = \{t_1, t_2, t_3, \dots, t_n\}$. A periodic task is defined by $t_i = (ph_i, r_i, c_i, d_i, p_i)$, where ph_i is the initial phase, r_i is the release time, c_i is the worst-case execution cycles (WCEC) required for execution of task t_i , d_i is the deadline, and p_i is the period. Tasks may have precedence relation between them. Let v and F be the two sets of discrete CPU supply voltage levels and frequencies respectively. Here, $|v| = |F|$ and $v_{ff} : v \rightarrow F$ (voltage-frequency function) an objective function that maps each voltage level to one and only one processor frequency which is maximum operating frequency in that supply voltage. In this example voltage/frequency levels that do not provide energy saving are not considered.

Assume the following task set $T = \{\tau_1 = (0, 0, 150 \times 10^6, 6, 13), \tau_2 = (0, 2, 50 \times 10^6, 3, 13), \tau_3 = (0, 0, 100 \times 10^6, 13, 13), \tau_4 = (0, 7, 60 \times 10^6, 9, 13)\}$. In addition to timing constraints, the specification contains the following relations: t_1 excludes t_2 , t_2 precedes t_3 , t_2 excludes t_1 , and t_2 precedes t_4 . For this example, the DVS platform adopted utilizes a Phillips LPC2106 processor, a 32-bit microcontroller with ARM7core.

More specifically, the CPU supply voltage/frequency levels adopted for this example are $v_{ff} = \{(1.04 \text{ V}, 20 \text{ MHz}), (1.07 \text{ V}, 30 \text{ MHz}), (1.26 \text{ V}, 50 \text{ MHz})\}$. Moreover, considering an average switching capacitance of 0.28 nF per clock cycle, the energy consumption is 0.45 nJ/cycle at 50 MHz, 0.34 nJ/cycle at 30 MHz, and 0.31 nJ/cycle at 20 MHz. These values were obtained from the following equation:

$$E = A_{sw} \cdot N \cdot V_{dd}^2$$

Where N is the number of cycles for execution of task t_i , A_{sw} is the average switching capacitance per clock cycle, and v_{dd} is the voltage supplied. A_{sw} is considered to be constant for all tasks in the example.

Figure (1.1) (a) shows a schedule obtained by adopting the optimal offline DVS algorithm proposed by (Yao et al., 1995), which relies on the optimal run time scheduling method, *earliest deadline first* (EDF). The schedule is invalid, since t_1 cannot be preempted by t_2 . As an alternative, Figure (1.1) (b) shows a schedule obtained by blocking the execution of t_2 while t_1 is executing. Again, the schedule is infeasible since t_2 misses its deadline due to the earlier release of t_1 . Even discarding DVS, in other words, running every task at the maximal voltage/frequency level, a schedule could not be found if EDF is adopted (or other runtime scheduling algorithm) (see Figure (1.1) (c)). On the other hand, Figure (1.1) (d) depicts a schedule found using the proposed pre-runtime method. All tasks meet their deadlines, and additional energy savings can be obtained comparing to a pre-runtime schedule without DVS as shown in Figure (1.1) (e).

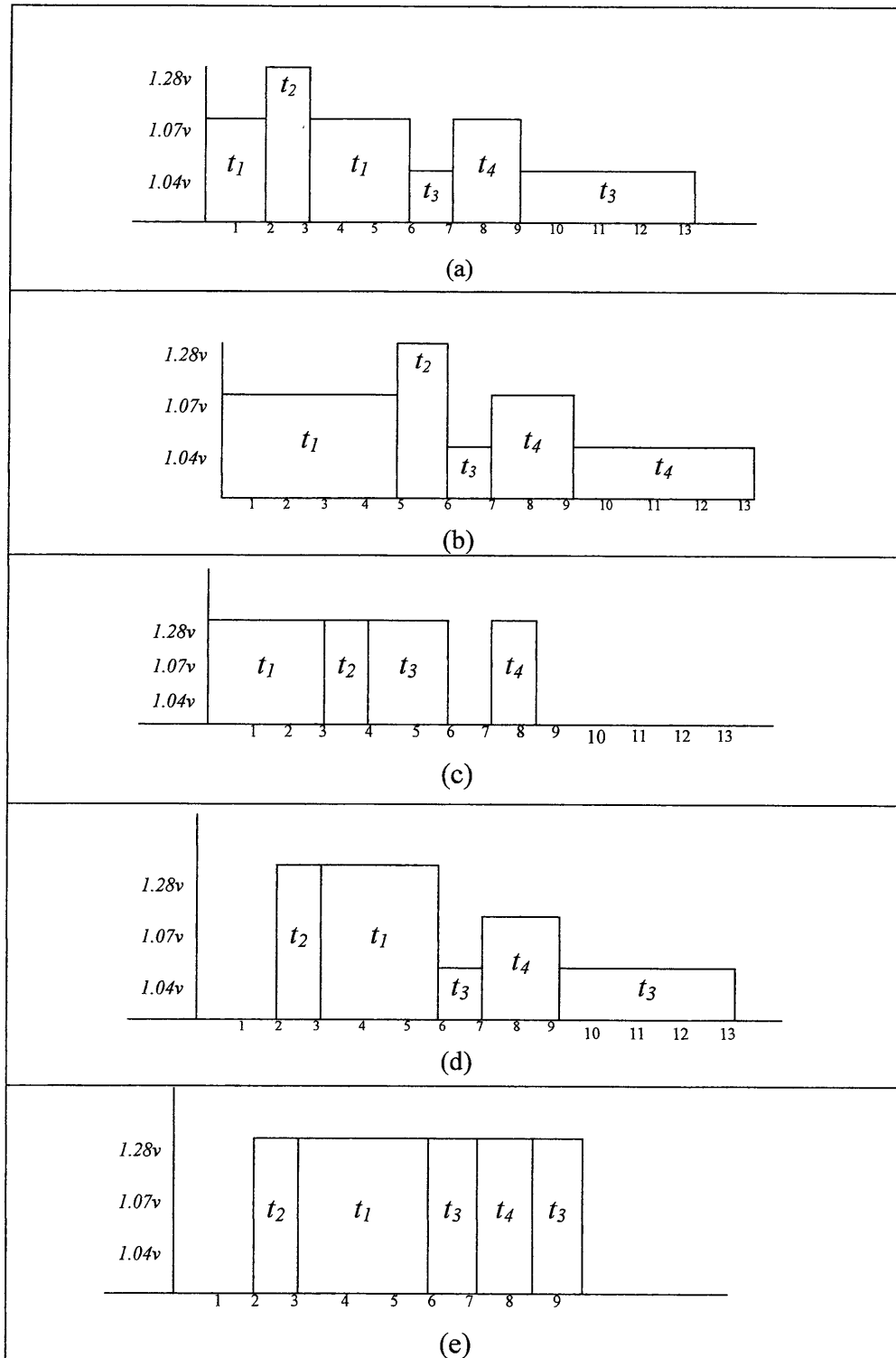


Figure (1.1)
Schedules generated according to different scheduling methods

Figure (1.1) (d) presents a schedule that consumes 0.1414 J, while the schedule depicted in Figure (1.1) (e) utilizes 0.162 J. Additionally, the processor needed to be left idle to find a valid schedule. In this example,

context and voltage/frequency switching overheads have not been taken into account and the CPU is assumed to have a halt instruction that avoids energy consumption on idle state. Henceforth, overheads are considered in this work.

1.7 Research Objectives

The following represent the main objectives of the research:

1. The objective of this research is to find an efficient solution for minimizing power consumed for scheduling real-time tasks on multi-processor system using dynamic voltage scaling.
2. Suitable iterative algorithm such as genetic algorithm and cuckoo search are to be employed to obtain efficient, or rather an optimal solution.
3. The capability of the adapted algorithms is investigated to achieve the aforementioned objective. Furthermore, a comparative analysis of the two algorithms is performed to judge as which of the two algorithms solves the underlying problem in a more efficient manner.

1.8 Methodology

The research methodology adopted in this work is shown in Figure (1.2). The first phase of the research is to build a mathematical model to define the problem formally. The model will define the problem in terms of objectives and constraints considering all assumptions of task model and power model. Then I pursue to adapt the algorithms to be used in the implementation phase. The two algorithms to be adapted are genetic and cuckoo search algorithms, which are implemented on basis of the specifications and task, power and system objective models (presented in Chapter 3) to test results empirically.

TGFF (Dick, Rhode & Wolf, 1998) is used to generate task graphs for experiments in this work. In the phase of tuning parameters, the sensitivity analysis will guide our experiments to select the best combination of parameters to be tested with the generated test cases.

The following stages will include scheduling tasks in different test cases using both selected algorithms. Results are to be analyzed extensively in terms of solutions quality and time efficiency. Best combinations of the adapted algorithms are then to be determined. Eventually results would be compared in terms of energy consumption rates and statistically tested to validate the results using Wilcoxon ranked-sum test at 95% confidence level using “Statistica5.1” package.

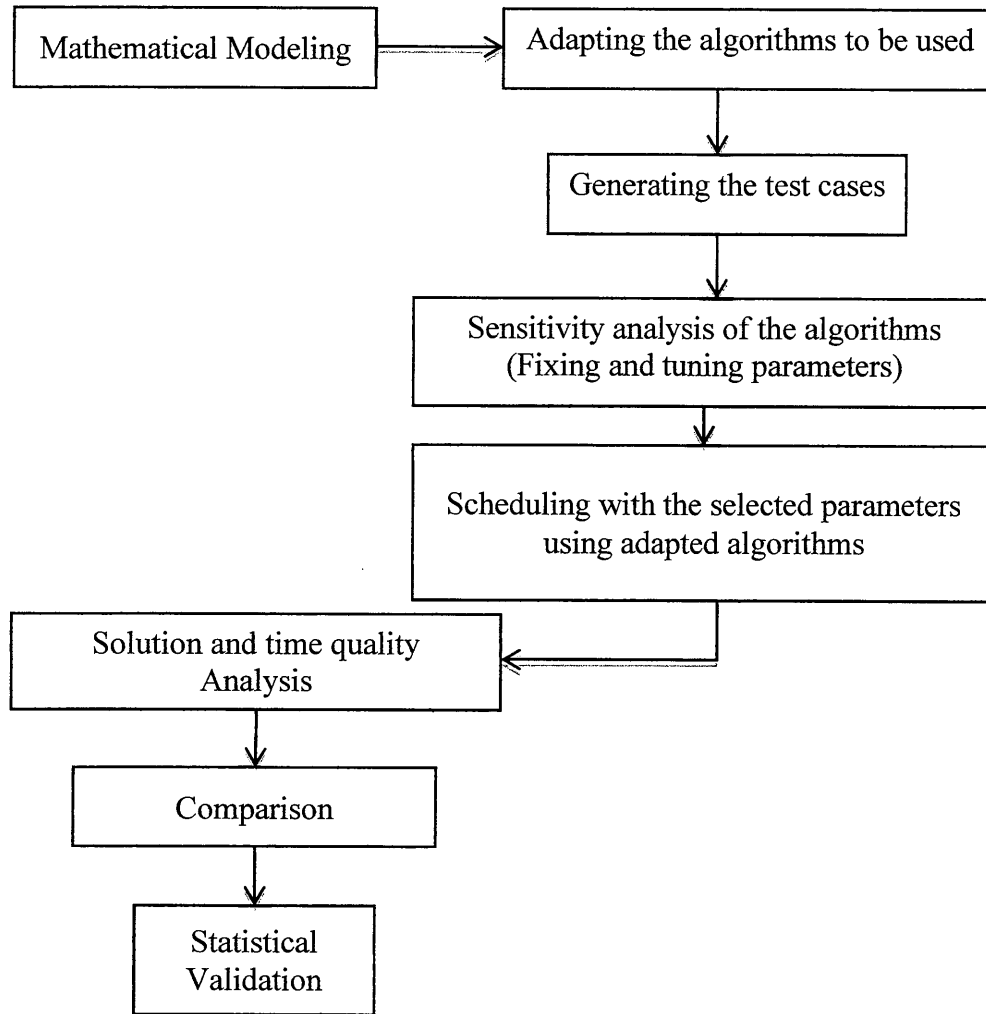


Figure (1. 2)
Study methodology

1.9 Research Contributions

The following summarizes the overall contribution of the research.

1. A suitable mathematical model of the problem is developed and presented in the thesis. The model considered the task dependencies, deadlines, and resource requirements.

2. Genetic algorithm and Cuckoo search algorithm (two population based algorithms) have been adapted to solve the optimization problem.
3. The effect of different parameters of GA and Cuckoo search on the solution quality has been investigated through an empirical study.
4. A comparative analysis of GA and Cuckoo search has been made in terms of quality of solutions and computational efficiency.

1.10 Thesis Outline

Chapter 1 provides an overview of the real-time systems, real-time task allocation and scheduling and dynamic voltage scaling. It also presents the problem statement and highlights the motivation. Literature review of previous studies is presented and discussed in Chapter 2. Chapter 3 explains the mathematical formulation of the problem and related task and power model specifications. In Chapter 4, the description of the proposed algorithms is provided. The experimental results, analysis and comparisons of the performance of proposed algorithms are presented in Chapter 5. Finally, the thesis concludes in Chapter 6.

Chapter two

Literature Review

2.1 Task Allocation in Distributed and Grid Systems	17
2.2 Task Allocation and Scheduling in Multiprocessor Systems	21
2.2.1 General Task Allocation Approaches	22
2.2.2 Real Time Task Allocation to Minimize Energy Consumption	23
2.3 A Review of Genetic Algorithm	35
2.4 A Review of Cuckoo Search	40
2.5 Previous Work Limitations and Gap Analysis	43
2.6 Summary	45

Chapter two

Literature Review

The previous chapter provided the basic knowledge and definition of the problem to be studied in this work. This chapter presents an overall review of the relevant work done on the field of energy aware scheduling of tasks on multi-processor systems, and more specifically, real time applications. Previous studies in similar fields are analyzed and discussed precisely to define their limitations and advantages.

The first section in the chapter presents task allocation on multi-processor environment in grid and distributed systems. This is followed by a discussion of task allocation techniques focusing on real time task allocation to minimize energy. Final sections present the adapted scheduling algorithms in the work, namely, genetic algorithm and cuckoo search.

2.1 Task Allocation in Distributed and Grid Systems

Task allocation is an important problem in the computational field and one of the topics that has been thoroughly studied in the area of computer science, mainly in the distributed and grid systems (Dogan & Ozguner, 2002). The problem is defined to be NP-complete problem (Golub & Kasapovic, 2002; Kousalya et al. 2008).

Steen (2003) defines distributed systems as a collection of computing devices that appear as single and coherent to end-users. Several inter-related factors affect the energy consumption in distributed

systems including task partitioning, process redundancy, fault tolerance, task and message scheduling, and communication bandwidth allocation (Prosperi et al., 2012). Processing streaming data efficiently using time and available resources is one of the major requirements in the computational field. Distributed computing is a promising approach to meet the growing computational requirements (Page & Naughton, 2005). Many forms of distributed architectures are developed to cope up with the requests and to efficiently perform task allocation to the available processing resources and routing data between them (Eidenbenz & Locher, 2016).

Grid computing systems with mobile nodes have been considered by Prosperi et al. (2012), who proposed a method to guarantee both lifetime and time constraints of grid applications. The study modeled the problem as a distributed energy constrained resources allocation, and proposed a price-based algorithm to provide the best solution (Prosperi et al., 2012).

Grid computing is considered as a type of distributed computing enabling sharing, selection, and aggregation of distributed resources across multiple domains depending on their availability, performance, and QoS requirements and makes them use for computational purpose (Foster et al., 2001; Foster & Kesselman, 2003). A computational grid provides an interface between the distributed resources and the user which is hiding all grid complexities from the user (Kousalya & Balasubramanie, 2009).

Heuristic techniques have been widely used by researchers to solve task allocation problem in grid computing. Grid scheduler checks workload and resource availability of each site and if any of the sites

satisfy job request then it is forwarded to that site for execution. Scheduling of jobs is done by grid scheduler and starting of job at site is done by local scheduler (Chandak et.al, 2011).

Khafa et al. (2008) used Memetic algorithm for grid task allocation for obtaining efficient batch schedulers for grid systems. Their simulation shows a good quality of dynamic scheduling that can be used to design efficient dynamic schedulers for real Grid systems. Liu et al. (2006) employed ant colony algorithm to model a solution for allocating tasks dynamically on grid systems. Chang et al. (2009) used a modified version of the ACO algorithm as they proposed a balanced ant colony optimization, trying to balance the entire load of processes on the system to minimize the makespan of a given set of jobs. Genetic algorithm is also adapted by some researchers to solve the problem of scheduling tasks in grid systems (Dai & Wang, 2006; Gao et al., 2005).

Other researchers employed meta-heuristics to solve the problem such as simulated annealing and tabu search. Fidanova (2006) used simulated annealing (SA) to achieve high throughput computing in a grid environment in which it guarantees good load balancing of the machines and it is applied in dynamic way. Fayad et al. (2007) implemented tabu search (TS) which uses fuzzy sets to model the uncertain processing time.

Dealing with real time systems (which are associated with multiple constraints such as deadline and possibly precedence of execution) on the basis of distributed architectures is a rich field prompting many researchers to investigate and try to enhance. Failure to meet the deadlines of critical real time tasks which may lead to serious failure of the system is one of the challenging problems that affect the overall performance of the network. Failure may result in specific agent/node

leaving the network, causing serious cost, such as network service disruption (Jin, Vural, Gluhak & Moessner, 2013; Prosperi et al., 2012; Tindell, Burns & Wellings, 1992).

Energy consumption during task allocation process in different stages was a major concern for many researchers. Zhu et al. (2007) developed an energy-aware and QoS-aware task allocation algorithm to optimize the energy consumption of applications submitted to a multi-hop cluster-based wireless sensor network. Their algorithm was serving to minimize the total energy consumption of the system under real-time constraint. The proposed approach saves energy at various steps, such as task mapping, routing, and task voltage scaling through using nested genetic algorithm to explore the solution space efficiently (Zhu et al., 2007).

The analytical model proposed by Mahapatra and Zhao (2005) for energy efficient scheduling in distributed real-time embedded systems to handle time-varying task inputs obtained significant energy saving when it was tested for real time multimode embedded systems. They adopted slack distributed scheme in scheduling task sets in the system. The scheme is made according to the demands of the nodes which affects the energy consumption in the system by determining the service rate and applies voltage scaling according to the dynamic traffic condition observed at various network nodes (Mahapatra & Zhao, 2005).

Prosperi et al. (2012) proposed two heuristic approaches to address the problem of allocating a task set and the required communication bandwidth on a distributed embedded system, aiming at reducing energy consumption while guaranteeing timing and redundancy constraints. Simulation results showed the effectiveness of the proposed approaches

and presented a systematic approach to address the problem of allocating a task set and the required communication bandwidth on a distributed embedded system, aiming at reducing energy consumption while guaranteeing timing and redundancy constraints. The research focused on distributed embedded to deal with the problem of dividing tasks and communication messages to achieve the goal of timing constraints while keeping overall energy consumption in minimum levels and maximizing task duplication.

2.2 Task Allocation and Scheduling in Multiprocessor Systems

Scheduling tasks on multiprocessor systems is considered NP-Hard problem (Chen et al., 2005). Different approaches have been proposed for multiprocessor scheduling trying to obtain a near-optimal solution of the problem (Rouhifar & Ravanmmehr, 2015). It is critical in multiprocessor scheduling to consider two main aspects, which are task ordering and scheduling, and voltage scaling (Xian et al., 2007). Thus, the scheduler has the dual responsibility of minimizing the execution time of the resulting schedule and balancing the load among the processors (Nedunchelian et al., 2006).

Mottaghi and Zarandi (2014) proposed a scheduling algorithm DFTS (Dynamic Fault Tolerant Scheduling) taking into consideration tolerance for transient faults. The algorithm schedule real time tasks according to tasks released time, the available processor, and rate and number of faults. The algorithm deals with the tasks upon its criticality by assigning a specific parameter to each task classifying them.

2.2.1 General Task Allocation Approaches

Different solutions were proposed by researchers to solve the problem efficiently taking in consideration the factors that best satisfy the requirements in different situations. Most approaches used by previous researchers to solve task allocation problem, such as graph theory algorithm to find the best route, were seeking to provide best solutions to maximize system throughput using least amount of the computational and resource requirements (Tindell et al., 1992). One of the widely used approaches due to the low complexity is heuristic-based techniques such as various list-scheduling heuristics, and greedy approaches which may only provide sub-optimal solutions. Guided random search techniques such as genetic algorithm can generate good quality of output schedules (Jin et al., 2013; Zhu et al., 2007).

Task partitioning is one technique used by many researchers to solve the problem of task scheduling on multiprocessor systems. AlEnawy and Aydin (2005) focused on periodic preemptive hard real-time systems where tasks are assigned rate-monotonic priorities whereas Chen and Theile (2008) proposed an approximation algorithm to partition tasks with various power consumption functions.

An ILP formulation for the energy aware model was proposed by Goraczko et al. (2008) which depends on computation of an optimal task partitioning for heterogeneous multiprocessors with duty-cycled interdependent real-time tasks. The model does not assume linear execution time or power consumption with respect to clock frequencies. It is based on a more generic notion of PSM with the cost of mode switching (Goraczko et al., 2008).

Zhu, Melhem and Childers (2003) introduced two novel power-aware scheduling algorithms for dependent and independent task sets based on the concept of slack sharing on multi-processor systems to reduce energy consumption. In both cases, they proved that scheduling with slack reclamation will not cause the execution of tasks to finish later than the completion time in canonical execution, where each task uses its worst case execution time. These algorithms proceed as reclaiming the unused time of execution of a task to reduce the execution speed of future tasks in order to reduce the total energy consumption of the system. The study considered the effect of discrete voltage/speed levels on the energy savings for multi-processor systems and simulation results show a considerable saving in energy consumption with variable voltage processors. It is worth mentioning that processors with a few discrete voltage/speed levels achieve approximately the same energy savings as processors with continuous voltage/speed, and the effect of voltage/speed adjustment overhead on the energy savings is relatively small (Zhu et al., 2003).

2.2.2 Real Time Task Allocation to Minimize Energy Consumption

In this section, I provide a review of a number of recent related research and approaches to the problem of scheduling of real time tasks in multiprocessor environment, focusing on the research that considers the aspect of energy consumption in their investigation and implementation.

Real-time systems are defined as those systems that must perform their processes correctly, considering the time constraints of system tasks (Buttazzo, 2011; Davis, 2014). Therefore, selecting the appropriate

algorithm which guarantees success of the system has a great effect on the overall performance. Important factors in real-time scheduling methods include issues such as response time, average of scheduling rate, energy consumption, and fault tolerance (Rouhifar & Ravanmmehr, 2015).

DVS is one of the approaches that have been used by several researchers to manage the problem of energy aware scheduling of real time tasks on multiprocessors. Tasks partitioning and dynamic scheduling are two main approaches studied extensively to solve the energy saving problem of scheduling real time tasks on a multi-processor system. In dynamic scheduling, tasks are to be selected by a global scheduler from a single ready queue then to be executed and transferred among processors (Xian et al., 2007).

Zhu et al. (2003) focus on energy ware scheduling for multi-processor real time systems. They used a run-time scheme depending on slack reclamation for tasks sharing a single and global deadline. Kadayif, Kandemir and Sezer (2002) considered the problem by determining the required number of processors to execute array-intensive applications. DVS scheme for chip multiprocessor was used by Juang et al. (2005).

Berl et al. (2010) reviewed methods and techniques used in the field of power consumption of computer hardware and network infrastructure in the areas of energy scheduling in multiprocessor and grid systems. The researcher identified some of the key challenges that arise in this field specifically, and also proposed cloud computing with visualization as a way to identify main sources of power consumption in systems and the significant trade-off between performance and QoS and energy efficiency. They concluded that the power saving can be achieved

in large scale computer services via implementing large scale hardware and software.

In case of real time systems, the scheduler has the dual responsibility of minimizing the execution time of the resulting schedule and balancing the load among the processors (Nedunchelian et al., 2006). For the purpose of scheduling real-time tasks in a system with the objective of reducing the consumption of energy while providing the strict probabilistic deadline guarantees, it is critical to consider two main aspects of task ordering, and scheduling and voltage scaling (Xian et al., 2007).

Tavares et al. (2008) used petri nets to provide a formal model to solve the problem of scheduling real-time tasks with strict deadlines (hard real-time systems) with voltage scaling. They considered the inter-task relations (e.g. voltage/ frequency switching) by introducing a pre-runtime method. The approach was used to provide a precise schedule generation, and allow property analysis and verification. Simulation results illustrate feasibility where it satisfies the objective of minimizing energy consumption as well as meeting system constraints. To guarantee meeting deadlines of all critical tasks they used pre-run time methods instead of run time scheduling methods for the reason that run time methods may fail in finding a feasible schedule (Tavares et al., 2008).

Maximizing energy saving on multiprocessor systems while satisfying the constraints of real time system is addressed by many researchers (Jaeyeon et al., 2010; Huang & Xu, 2010; Zhang et al., 2002; Zhang et al. 2005; Kumar, Manimaran & Wang, 2009; Swaminathan & Chakrabarty, 2001). Huang & Xu (2010) proposed a technique which is based on identifying set of “good” task allocation and schedules for

execution in sense of energy consumption in a chip. They presented the task allocation (resource binding) with binary representation to formulate their solution focusing on the embedded systems in the design. They concluded that the basic issue in designing such systems is to build scheduling algorithm for task allocation to use the available processors effectively with multiple constraints such as timing constraint.

Jaeyeon et al. (2010) presented a novel dynamic scheduling algorithm for re-allocating the slack for future tasks to reduce energy and satisfy deadline constraint on parallel machines. The proposed technique depends on the idea of DVS in which each task is assigned with different voltage. Their work focused on parallel and distributed systems. Their technique was used on real time systems and imposes the need to find an upper bound or the worst case in the actual execution to guarantee that all tasks are completed within the time limit. Their method shows effective execution time when tasks are overestimated or underestimated and the experiments show that their method offers improvement over the simplistic greedy method in terms of energy requirement but in regard to the time requirement the greedy algorithm is better. The application model used the assignment algorithm of tasks in a directed acyclic graph (DAG) to the appropriate processors with considering two cases related to the actual execution time (AET) and the estimated execution time (ET). The first case is when the $AET < ET$ and the second case is when the $AET > ET$.

The scheduling algorithm proposed by Kang and Ranka (2010) considered reallocating the slack for future tasks to reduce energy and/or satisfy deadline constraints. Their approach is comparable to static algorithms applied at runtime in terms of energy minimization and

deadline satisfaction, but requires considerably smaller computational overhead. Their algorithms work better in such environments where the estimated time for all tasks is greater than or equal to the execution time (underestimation) or where the estimated time for all tasks is less than or equal to the execution time (overestimation). Furthermore, their approach was equally effective for hybrid environments where some tasks complete before estimated time while some tasks complete after estimated time (Jaeyeon et al., 2010).

Zhang et al. (2002) presented a frame work which integrates the voltage selection of multiple processors of given different voltages and scheduling tasks to minimize energy consumption. Zhang et al. (2002) and Kumar et al. (2009) employed earliest deadline first algorithm to design a solution for the problem on multiprocessor system. The approach used by Zhang et al. (2002) is considering the priority of tasks and best fit processor assignment for each task. They used integer linear programming in formulating the voltage selection problem on a DAG of pre-ordered tasks. They have considered the transition overhead in the calculation of energy consumption and voltage scaling calculations (transition overhead is when a processor changes the voltage level from one to another). It can be completed optimally in polynomial time for continuous voltage and some special cases of discrete voltage or solved efficiently by their approximation for general discrete voltages.

Since power consumption is a very important aspect in designing any system, Jejurikar and Gupta (2006) dedicated their work to find the optimal solution to minimize the run-time energy consumption of the system and to provide a mathematical explanation of how the supply voltage decrease is related directly (proportionally) to decreasing the

power consumption of the processor. They proposed the stack based slowdown algorithm based on optimal feasibility test and presented a dynamic slack reclamation algorithm that work in conjunction with it, and it resulted in higher energy efficiency. The algorithm designed by the group of researchers was based on centralized scheduling policy (can be enhanced to be based on more distributed environment).

Jejurikar and Gupta (2006) employed the EDF scheduling technique to address the slowdown factors issue for non-preemptive task system by presenting a stack based slowdown algorithm. They proposed a dynamic slack reclamation policy for further enhance energy savings. Similarly to Tavares et al. (2008) in which they also proposed a technique that considers the dynamic slack times to allow more reduction in energy during system execution. Simulation results of their study shows that the energy gains using static slowdown factors compared to dynamic slowdown is better since the first resulted in 15% compared to 20% in the second approach over the known slowdown techniques.

MORA is a slack reclamation design proposed by Nelis and Goossens (2009) aiming to reduce the energy consumption while scheduling a set of sporadic constrained-deadline tasks by a global, preemptive and FJP algorithm on a fixed number of DVFS-identical processors. The proposed algorithm MORA take into consideration the application specific consumption profile of the tasks and exploits early job completions at run-time by starting the execution of the next waiting jobs at a lower speed in order to improve the energy saving. Simulation results shows that it can save up to 32% of energy (on average) compared to execution without using any energy-aware algorithm (Nelis & Goossens, 2009). Jejurikar et al. (2006) focused on slowdown factors in

their proposed solution to be computed and considered to minimize the energy consumption for embedded real time systems where tasks arrive periodically with specific fixed deadlines.

The solution proposed by Shin and Kim (2003) is focusing on minimizing energy consumption for multiprocessor systems which operate on DVS for based real time systems. It is composed of two main stages; the first is to decide the order of execution where the second step is to determine execution speed (task stretching). Experimental results show that their proposed technique is reducing power by 50% on average over non-DVS task scheduling algorithms. Their method can work on conditional task graphs which model more complex precedence constraints. They first propose the condition unaware task scheduling by integrating the task ordering algorithm for conditional task graph and the task stretching algorithm for unconditional task graphs. They also propose condition aware task scheduling algorithm and the profile aware task scheduling algorithm by considering run time behavior and the profile information of the conditional task graphs (Shin & Kim, 2003).

Xian et al. (2007) applied a variation of the worst-fit decreasing bin packing algorithm for balancing load with respect to a mathematical parameter related to expected energy consumption in their method. They improved task partitioning approach for its known simplicity and ease of implementation in which the tasks are assigned to a particular processor permanently. The study took advantage of the probabilistic workload information. In contrast to all previous studies, the study provided a mathematical formulation of the problem focusing on uncertain workloads under hard real-time constraints (they consider sets of periodic real-time tasks processed on a set of identical processors, the energy model

they used assumed that the processors has unbounded and continuous range of frequencies to choose from). The researchers designed a polynomial-time heuristic method which is earliest deadline first (EDF) scheduling. This was to be converted to a probability based load balancing and solved with worst-fit decreasing bin-packing heuristic to balance the load where they use execution cycles to express the workload instead of execution time (execution time is the execution cycles of a specific task divided by the processor's frequency). Their simulation results shows better energy saving compared to existing methods with synthetic, multimedia, and stereo-vision tasks.

The proposed scheduler proposed by Bautista et al. (2008) granted a saving in energy from 34% to 74% while satisfying the constraints of soft real-time applications on high-performance embedded systems (i.e., multithreaded and multi-core) over some other existing scheduling methods. The used approach pursues to reduce the number of DVS switches by changing the voltage and frequency of all the cores at the same time (increase the frequency in case to satisfy real-time task requirements or reducing the frequency if a task's computation time can be guaranteed). Power savings have been explored in three different scenarios (varying the inter-arrival time) and experimental results show that, on average, savings provided by the proposed algorithm are about 34%, 18%, and 67% across the scenarios (Bautista et al., 2008).

PreDVS technique proposed by Wang and Mishra (2010) is different of existing inter- or intra task DVS techniques that the processor voltage level is adjusted multiple times throughout each task's execution to potentially attain more energy savings. However it can be powerfully engaged with existing intra-task scaling techniques that this technique can

be considered as a complementary to existing intra-task DVS techniques. Furthermore, any intra-task scaling can be applied after PreDVS to further reduce energy consumption at runtime. It exploits static time slack at a finer granularity and energy reduction achieved is more than the saving in inter-task scaling without introducing any extra voltage switching overhead. It provides a fully polynomial approximation scheme which can guarantee to give solutions within specified quality bounds that the total energy consumption can be minimized while no task deadline is violated, specially that the focus of the study is on hard real-time systems with preemptive periodic task sets (Wang and Mishra, 2010) .

The approach proposed by Chen et al. (2008) pursues to explore task partition and scheduling for the minimization of energy consumption in homogeneous multiprocessor systems through applying DVS. The method gives a polynomial-time approximation solution for the problem which is considered as a strongly NP-hard problem to provide worst case guarantees in the expected energy consumption of the derived solutions (Chen et al., 2008).

AlEnawy et al. (2005) proposed an algorithm that is based on reserving a subset of processors for light tasks to guarantee a consistent performance. Also Jejurikar and Gupta (2006) used DVS based on computation of static slowdown factors which can lead to significant saving in energy consumption on a real-time system where tasks arrive periodically and have deadlines. Their approach used the computation of task slow down factors to guarantee meeting all task deadlines constraints. They studied the effect of static and dynamic priority scheduling policies and have considered computing slowdown factors under rate monotonic (RM) scheduling and earliest deadline first (EDF)

scheduling, where the former (RM) is showing that it has pseudo polynomial time complexity later (EDF) proved to have polynomial time complexity (Jejurikar & Gupta, 2006).

Kumar et al. (2009) presented heuristic scheduling algorithms to minimize the consumption of energy while guaranteeing the system constraints. They have tested their approach for scheduling real-time messages over a shared-medium on wireless network. They used hop-by-hop transmission in their technique instead of high energy experiencing direct hop transmissions as a basis to solve this problem. Two energy-aware scheduling algorithms were proposed and compared them against ILP solution of the scheduling problem. The simulation results proved that the proposed algorithms achieve a comparable saving in energy to that obtained using the linear programming methodology under practical channel conditions.

Tavares et al. (2008) presented the solution based on a time Petri nets for power-aware hard real-time systems schedule generation, considering dynamic voltage scaling, overheads, precedence and exclusion relations. They induced several experiments to determine the feasibility of the proposed method (Tavares et al., 2008).

Alahmad and Gopalakrishnan (2011) considered the problem of scheduling of periodic and independent real time tasks on heterogeneous processors with discrete voltage level to fulfill the main objective of minimizing the power consumption with guaranteeing the satisfaction of system constraints. Two important issues were taken into consideration in the design; energy consumption and processor heterogeneity. However, since the problem is NP-hard, the study presented a fully polynomial time approximation scheme for it. Energy consumption of the system is

dependent on the decisions regarding task allocation and speed settings (Alahmad & Gopalakrishnan, 2011).

Kargahi and Movaghar (2008) was focusing their study on real time tasks which arrive periodically and have exponentially distributed service times and relative deadlines. The approach designed is aiming to manage power consumed of the system while guaranteeing an upper bound on the fraction of jobs missing their deadlines. Their design can be applied to non-Markovian systems also using a hybrid analytical-simulation method. Experiments show improvement in the tradeoff between power and performance on the average service time (Kargahi & Movaghar, 2008).

Zhu and Meuller (2004) proposed a novel scheduling approach design combined DVS with feedback control schemes in order to reduce energy consumption significantly. The scheme extends EDF for hard real-time systems with dynamically different fluctuating workload characteristics in a most aggressive manner. This enables the system of selecting the suitable frequency and voltage settings. The design depends on the operating system to implement both the real-time scheduler and the feedback controller. The complexity of the algorithm is $O(n)$ for n tasks. The feedback technique makes the system capable to select the right frequency and voltage settings. This significantly reduced that energy consumption while guaranteeing the timing constraints of hard real-time tasks, so that no tasks ever miss their deadlines. For predictable fluctuating execution time patterns, their feedback DVS scheme is able to adapt to dynamically fluctuating workloads better than previous work and saves up to 29% additional energy. The scheme is not sensitive to any particular workload characteristics, i.e. the execution time patterns, and is

capable of scaling for task sets with different number of tasks. As future work, they proposed to include the assessment of the algorithm under real embedded environments and the investigation of the impact that different PID parameters have together with systematic approaches in parameter tuning.

Swaminathan and Chakrabarty (2001) proved that the minimum-energy scheduling problem is NP-complete and presented algorithms that generate near-optimal solutions for task sets of varying size in polynomial time. They showed that mixed-integer linear programming (MILP) is a tool that may be used for exact solution of moderate-sized problems. The proposed algorithm had been tested on multiple real-time task sets and results showed that the algorithm provides near-optimal solutions.

In order to achieve the goal of enabling real-time applications containing complex runtime behaviors to be efficiently supported on multiprocessors, the experimentation by Liu (2013) designed new multiprocessor real-time scheduling algorithms and efficient schedulability tests for tasks that may contain self-suspensions, graph-based precedence constraints, non-preemptive sections, and parallel execution segments. Another goal is avoiding over-provisioning systems and to reduce the number of needed hardware components to the extent possible while providing temporal correctness guarantees (Liu, 2013).

Bat intelligence is another heuristic which was used by Malakooti et al. (2013) to solve the problem of reducing energy aware multiprocessor scheduling subject to multiple objective which are makespan, and tardiness. Their simulation results show a considerable improvement over genetic algorithm in terms of solution quality. The simulation results of the bi-objective problem show a clear conflicting

between make span and energy and tardiness and energy. As they mentioned, their work was the first in the field to relate the three objectives where no other prior research has previously shown it (Malakooti et al., 2013).

2.3 A Review of Genetic Algorithm

GA is a search algorithm that iteratively transforms a set of strings, each with an associated fitness value, called a population, into a new population of offspring objects using the Darwinian principle of natural selection and using operations such as crossover and mutation. Solutions from one population are taken and used to form a new population. Genetic algorithms were proven to be the most powerful optimization technique in a large solution space (Tsang & Au, 1996).

GA is used by large companies to optimize schedules and design products that range from large aircraft to tiny computer chips to medicines. The Los Alamos National Laboratory has used them to analyze the data from satellites (Mitchell, 1998; Carr, 2014).

GA encodes the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The strings which are candidate solutions to the search problem are referred to as chromosomes, the alphabets are referred to as genes and the values of genes are called alleles. To evolve good solutions and to implement natural selection, we need a measure for distinguishing good solutions from bad solutions. The measure could be an objective function that is a mathematical model or a computer simulation, or it can be a subjective function where humans choose better solutions over worse ones. Genetic algorithms rely on a

population of candidate solutions. The population size, which is usually a user-specified parameter, is one of the important factors affecting the scalability and performance of genetic algorithms. For example, small population sizes might lead to premature convergence and yield substandard solutions. On the other hand, large population sizes lead to unnecessary expenditure of valuable computational time. Genetic Algorithms aid to look for the best solution among a number of possible solutions (Sastry et al., 2005).

GA is used to solve single objective multiprocessor scheduling problem using make span, tardiness, and energy consumption as objective function. Different combinations of these objectives are used to solve bi-objective multiprocessor scheduling problems such as make span vs. energy and tardiness vs. energy (Malakooti et al., 2013). GA has been used by many researchers as an efficient method to solve the problem of task scheduling. Some researchers used the pure version of GA to schedule tasks in multiprocessor system (Miao et al., 2008). The advantages of this algorithm are that it is simple to use, require minimal problem specific information and is able to effectively adapt in dynamically changing environment. The primary disadvantage of GA algorithm is that it has a long execution time (Wu, Yu, Jin, Lin & Schiavone, 2004).

Jin et al. (2012) adapted the idea of using the genetic algorithms to provide guarantees for real time problems. Their solution provided a semi-optimal solution which efficiently dealt with the parallel processing and effectively use resources in cost effective communications to improve the power consuming in multi-hop wireless networks. A tradeoff between the collaborative nodes and keeping the deadlines in the system were

considered by adding some modifications to design a hybrid theme of the fitness function to be embedded in the algorithm since GA is time consuming function and will efficiently work directly with networks with dynamic conditions and topologies or distributed systems. The solution included dividing the execution of GA into sequential stages, where each stage 'would require less resource and can be executed more quickly (Jin et al., 2013; Yichao et al., 2012).

Miao et al. (2008) proposed a novel coding scheme of multi-objective hybrid GA to for the purpose of energy saving task scheduling on chip multi-processor. The method they used is able to describe both the information of task allocating and operating voltage. The simulation they testing proved their hypothesis that the efficiency of task scheduling on CMP increased while execution time is decreased along with reducing the system consumption of power for 40-46%; that they improved the basic operators which are crossover and mutation operators of the GA.

Nedunchelian et al. (2006) employed genetic algorithm in solving the problem of minimizing energy consumption. The initial population is generated randomly and the algorithm is using a straightforward encoding scheme. The fitness function depends on the balance of load among the processors, the max span, and the communication costs while evaluating the schedules. The selection function used is an extension of the roulette wheel selection where it is a stochastic sampling with partial replacement selection in order to increase the possibility of survival of the fitter solutions. To avoid getting stuck in a local maximum, random swapping was performed. Cycle crossover preserves the characteristics of the parent chromosomes in the children there by leading to exploration of search space (Nedunchelian et al., 2006).

Another study used a modified version of genetic algorithm by Yi-Lee and Chen (2003), to schedule parallel program on multiprocessor system and constructed a simulation and evaluation environment to evaluate it. Scheduling goal is to find a schedule that minimizes the entire makespan.

The GA algorithm is adapted by Lin et al. (2005) aiming to reduce the energy consumption related to static scheduling of real-time tasks on dynamic voltage supply processors, while guaranteeing the constraints of the tasks. Deadline and precedence constraints are important factors to be considered in the design of the algorithms in this study to work efficiently on heterogeneous processors. Experimental results show that the employed approach results in reducing energy consumption in a range from 20% to 90%. Compared with two other list scheduling algorithms and simulated annealing, also in regard to solution quality it performing better for most scheduling problems.

Wu et al. (2004) proposed GA approach for the problem of task scheduling for multiprocessor systems. Key features of this approach include a flexible, adaptive problem representation and an incremental fitness function. Kamiura et al. (2002) developed a parallel genetic algorithm for multi-objective optimization problems (MOGADES) which can derive widespread Pareto optimal solutions. Page, Keane, and Naughton (2010) developed a scheduling algorithm to schedule heterogeneous processors in a distributed computing system. This algorithm provides efficiently schedules and adapts to varying resources availability. Also, it consistently uses processors more efficiently than the current state-of-the-art GA algorithms for the same problems. The most important drawback of this algorithm is that it was not tested under the

real-world condition and the efficiency of the algorithm for time critical applications have not been studied. Zomaya et al. (1999), presented a scheduling algorithm based on genetic algorithms to efficiently solve the scheduling problem without the need to apply any restricted assumptions that are problem-specific such is the case when using heuristics. Tripathi et al. (2000) developed a method based on genetic algorithms to allocating multiple tasks on heterogeneous distributed computing system taking into account the execution and communication costs.

Golub and Kasapovic (2001) developed an efficient genetic algorithm for scheduling precedence constrained task graphs without taking into account the communication cost. This algorithm was based on the task priorities. It selects the task with highest priority first. If more than one task has the same priority the task is selected randomly.

Lei et al. (2008) proposed a novel coding scheme of multi-objective hybrid GA for the purpose of energy saving task scheduling on chip multi-processor. The method they used is able to describe both the information of task allocating and operating voltage. The simulation they tested proved their hypothesis that the efficiency of task scheduling on CMP increased while execution time is decreased along with reducing the system consumption of power for 40-46%. They improved the basic operators which are crossover and mutation operators of the GA (Lei et al., 2008).

Samal, Mall and Tripathy (2014) have proposed a heuristic approach named GFTS (Genetic Algorithm Based Fault Tolerant Scheduler) for fault tolerant scheduling of aperiodic and hard real-time tasks on multiprocessor systems using genetic algorithm (GA). The scheduling algorithm is proposed based on GA and is combined with

information about the scheduling of real-time tasks to provide fault tolerance in multiprocessor environments. It uses PBFTS (Primary Backup Fault Tolerant Scheduling) with novel form of GA. GFTS acts better than previous methods for fault tolerant scheduling with the primary-backup in terms of system efficiency and performance.

2.4 A Review of Cuckoo Search

Cuckoo Search (CS) is a nature-inspired iterative optimization algorithm proposed by Yang and Deb (2009). The idea behind this algorithm was inspired by the breeding behavior of cuckoo in which the bird lays its eggs in the nests of other host species. In the case that the host bird discovers the eggs are not of their own, it either throws them away or abandons its nest to build another one elsewhere.

The algorithm was proposed as a tool for numerical function optimization and continuous problems. The algorithm has been applied to engineering optimization, image processing, scheduling, planning, feature selection, forecasting, and real-world applications because of its simplicity, efficiency, and promising results (Fister et al., 2014; Yasar, 2016). Yasar (2016) highlighted the superiority of CS algorithm due to the fact that it involves less number of parameters that need to be fine-tuned compared to most other similar algorithms.

The original CS algorithm evolves from the following three behavioral patterns of real cuckoos (Mohamed, Zain, Bazin and Udin, 2013; Yasar, 2016)

- Each cuckoo lays one egg at a time. The egg is dumped in a nest randomly chosen by the cuckoo.

- The best nests with high quality of eggs (solutions) will carry over to the next generations.
- The number of available host nests is fixed, and a host can discover an alien egg with probability $p_a \in (0,1)$. In this situation, the host bird can either throw the egg out of its nest or abandon the nest in order to build a completely new nest in a new location.

Each nest represents a potential solution in the search space. The CS algorithm also determines how to update the position of an egg laid by a cuckoo. Each cuckoo updates its position of laying egg based on current step size via Lévy flights. Lévy flight is a natural phenomenon noticed in some birds and fruit flies. It is a combination of short and very long steps, with sudden turns. These sudden turns are of essential importance for the CS algorithm, and determine the next position of the bird/fly using the following equation:

$$x_i(t + 1) = x_i(t) + \alpha * Levy(\lambda)$$

where $\alpha > 0$ represents a step size. This step size should be closely related to the scale of the test function that the algorithm is applied on. In most cases, α can be set to the value of 1 (Yang and Deb, 2009). It has been shown that the use of Lévy flight is much more efficient in exploring the search space as its step length is significantly longer when a large number of steps are performed compared to a simple random walk (Walton et al., 2011). The random step length is drawn from a Lévy distribution which has an infinite variance with an infinite mean:

$$Levy \sim u = t^{-\lambda}, \lambda \in (0, 3]$$

The consecutive positions generated through steps/iterations of a cuckoo create a random walk process which obeys a power-law step length distribution with a heavy tail. The original description of the CS algorithm as designed by Yang and Deb (2009) is represented in Figure (2.1).

```

Initialize a population of n host nests,  $i = 1, 2, \dots, n$ 
For all x
for all  $x_i$  do
do
    Calculate fitness  $F_i = f(x_i)$ 
End for
While Number Objective Evaluations < Max Number Evaluations do
    Generate a cuckoo egg ( $x_j$ ) by taking a Lévy flight from random
    nest
     $F_j = f(x_j)$ 
    Choose a random nest  $i$ 
    If ( $F_j > F_i$ ) then
         $x_i \leftarrow x_j$ 
         $F_i \leftarrow F_j$ 
End if
    Abandon a fraction  $p_a$  of the worst nests
    Build new nests at new locations via Lévy flights to replace
    nests lost
    Evaluate fitness of new nests and rank all solutions
end while

```

Figure (2.1)
Cuckoo search algorithm

Many variations of the algorithm were proposed by researchers for speeding up the convergence of the results and for better fine-tuning of the solutions (Ouyang, Pan, Yue & Du, 2014).

Walton et al. (2011) proposed a strong objective function in the Modified Cuckoo Search (MCS). They modified the algorithm producing high convergence rate (where it was able to outperform other optimizers) of the results in order to reduce the number of objective function evaluations required to find the global minimum.

Binary Cuckoo Search (BCS) was designed by Rodrigues et al. (2013) for the purpose of developing a selection called BCS in a search space modeled as a number of features-cube. Each nest in the algorithm is attached to a set of binary coordinates defining a feature relation to the final set of features or not, and the function to be maximized is the one given by a supervised classifier's accuracy.

Improved Cuckoo Search (ICS) is a modified version of the original CS which is using a fixed value for both p_a and α . Valian et al. (2011) presented a new approach of using variable values of the parameters which can be changed dynamically with the number of generations. They mentioned that the key to improve the performance of CS algorithm lies with the way of adjusting p_a and α . The algorithm is then employed for training feed forward neural networks for two benchmark classification problems (Valian et al., 2011).

2.5 Previous work Limitations and Gap Analysis

Various previous studies on RT methods focused on independent tasks without considering dependencies between tasks (Alahmad and Gopalakrishnan, 2011; Mohammedi, 2009; Golub and Kasapovic, 2001). In this study we consider dependencies in the proposed model of solution.

Oh and Wu (2004) presented a multi-objective GA scheduling non-preemptive tasks in soft real time system with multiprocessor. This algorithm didn't refer to the conflict between objectives and assume the performance of all processors the same. Page and Naughton (2005) presented GA for dynamic scheduling on heterogeneous multi-processor system. But all these algorithms are designed for normal tasks without

time constraints. Other adaptations of GA also didn't consider the initial population in all generations. Devi et al. (2014) used genetic algorithm for multiprocessor scheduling for dependent tasks. They have designed the selection operator using the crowded comparison operator to select best solutions. In our research real time tasks are to be allocated on homogeneous multiprocessor system, and time and dependency constraints are considered. Roulette wheel and elitism techniques are deployed to design two variations of the natural selection of the adapted GA algorithm considered in this study.

The standard cuckoo search algorithm was proposed with fixed values of λ and p_a (Yang & Deb, 2009). In our adaptation we will test multiple values of the parameters to ensure using the best combination for scheduling different test cases on multiprocessor system.

The values of λ and p_a parameters used in ICS proposed by Valian et al. (2011) are changing dynamically with the number of generations. However, the effect of the change of parameter values is studied herein to determine the best values to be used to enhance the performance of the algorithm.

The CS was originally developed for continuous domains, however many problems are defined in discrete domains. A discrete binary encoding of cuckoo search was used by Zheng et al. (2012) to solve knapsack problem in discrete domain. Ouyang et al. (2013) proposed a discrete cuckoo search for solving spherical Traveling Salesman Problem using a similar value of the abandon fraction p_a but the method used for search for a new nest operator of their algorithm is different from the one of basic CS. In this research a discrete integer encoding is used to solve the allocation problem of real time tasks on multiprocessor system. The

adaptation of the CS algorithm proposed method for determining suitable value for its parameters (i.e. λ , p_a) which is different than the values used in the original version of the algorithm by Yang and Deb (2010).

The population considered in all variations of Cuckoo search includes only valid solutions. However in this study, the proposed variation of basic CS (called CS with penalty) designed to allow the population to include some invalid solutions of specific percentage of the population with added penalty.

2.6 Summary

Energy aware scheduling of real time task is a rich field for research which captured the attention of many researchers trying to find optimal or sub-optimal solutions using different approaches and deploying various types of heuristics. In this, I have reviewed the relevant literature and provided a background on genetic algorithms and cuckoo search, the two algorithms employed in the underlying study.

Chapter Three

Problem Formulation

3.1	Notations	47
3.2	System Models	48
3.2.1	Task Model	48
3.2.2	Power Model	51
3.3	Objective Function and Constraints	55
3.4	Summary	56

Chapter Three

Problem Formulation

The previous chapters provided an overview of the problem of task scheduling on multi-processor systems and discussed and highlighted the core aspects of the earlier research studies in this field. This chapter presents the mathematical formulation of the problem and specifically describes the objective function and related constraints.

3.1 Notations

The notations used in this chapter and rest of the thesis are given in Table (3.1).

Table (3.1)
Notations

m	number of processors
n	number of tasks
v_i	Supply voltage provided to execute task t_i .
v_t	Threshold voltage
e_i	Energy consumed for processing task t_i
c_{eff}	Effective switching capacitance, where it is considered to be constant for all tasks
l_k	Number of discrete voltage levels on which processor p_j can operate on
f_i	Clock frequency at run time (operational frequency)
T	The set of all tasks; $T = \{t_1, t_2, \dots t_n\}$.
T_i	A Task with identification number i
P	The set of Processors; $P = \{p_1, p_2, \dots p_m\}$
p_j	Processor on which a task is mapped.
T_i	Execution time of task t_i
c_i	Number of clock cycles required to process task t_i .
pow_i	Power consumption of executing task t_i
e_{ijk}	energy consumed of processing task t_i on processor p_j at

	voltage v_k and frequency f_i
E	Total energy consumed to process all the n tasks
x_{ijk}	$X = \{x_{ijk} \in \{1,0\} i \in [1,n], j \in [1,m], k \in [1,l_k]\}$ Any element of X : $x_{ijk}=1$ if a task t_i is allocated to processor p_j operating on voltage level v_k , otherwise $x_{ijk}=0$.
EST_i	Earliest start time for task t_i
LST_i	Latest start time of task t_i
FT_i	Finish time of task t_i
$Pre(t_i)$	Set of predecessors of task t_i (precedence tasks)
$Succ(t_i)$	Set of immediate successors of task t_i
ST_i	Actual start time of task t_i
d_i	Deadline of task t_i
VP_{p_m}	Set of voltage levels of processors in the multiprocessor system. $VP_{p_m} = \{VP_{p_1}, VP_{p_2}, \dots, VP_{p_m}\}$ Where m is number of processors
VP_{p_j}	Set of voltage levels that a processor p_j operates on. $VP_{p_j} = \{VL_1, VL_2, \dots, VL_{l_k}\}$ Where l_k is number of voltage levels a processor p_j operates on.
VL_k	Each level represents the voltage and frequency pair of the processor p_j at a specific voltage level VL_k . $VL_k = (v_j, f_j)$; where v_j is the voltage supply of processor p_j at voltage level VL_k and f_j is the frequency of the processor p_j at voltage level VL_k . $1 \leq k \leq l_k$

3.2 System Models

3.2.1 Task Model

Assume having a job consisting of a set of n real-time tasks $T = \{t_1, t_2, \dots, t_n\}$. Each task t_i has a workload denoted by c_i which represents the computation requirement (number of cycles) of the task and a deadline denoted by d_i . We also assume that the number of clock cycles (workload) for each task is deterministic and is known *a priori* and tasks are non-preemptive and hence cannot be interrupted during their execution. A task may communicate with other tasks and hence have

precedence relationships. The precedence relationship between tasks is presented by a directed acyclic graph (DAG) in which tasks are represented by nodes and the dependency between tasks are represented by edges. A typical task graph is shown in Figure (3.1).

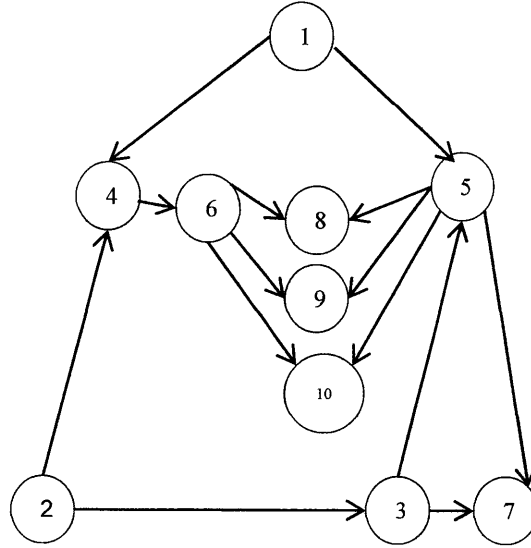


Figure (3.1)

A task graph with 10 tasks

Definition 3.1. $Succ(t_i)$ denotes the set of immediate successors of task t_i .

In Figure 3.1, $Succ(t_4) = \{t_6\}$ and $Succ(t_5) = \{t_7, t_8, t_9, t_{10}\}$.

Definition 3.2. $Pre(t_i)$ is the set of immediate predecessors of task t_i .

In Figure 3.1, $Pre(t_4) = \{t_1, t_2\}$ and $Pre(t_5) = \{t_1, t_3\}$.

Definition 3.3. If a task t_i has the set of predecessors $Pre(t_i)$ then t_i cannot start executing on a processor unless all of its predecessor tasks are completed.

Definition 3.4. Total processing time, τ_i , of a task t_i at frequency f_i is given by:

$$\tau_i = c_i / f_i \quad (3.1)$$

Where c_i is the number of clock cycles to execute task t_i and f_i is the processor operating frequency.

Definition 3.5. Latest start time of a task t_i is the time that if the task didn't start by then it will miss its deadline. The latest start time of a task t_i is given by:

$$LST_i = d_i - \tau_i \quad (3.2)$$

Where d_i is the deadline of task t_i .

Definition 3.6. If a task does not start by its latest start time then it will miss its deadline.

Definition 3.7. The earliest start time EST_i of a task t_i is given by:

$$EST_i = \begin{cases} \max_{t_j \in Pre(t_i)} \{FT_j\} \\ 0 \text{ if } Pre(t_i) = \emptyset \end{cases} \quad (3.3)$$

Where $pre(t_i)$ is set of predecessors of task t_i (precedence tasks), FT_j is the finish time of task t_j . Task t_j is a predecessor of task t_i with the maximum finishing time.

Latest start time and earliest start time are two times used in the scheduling algorithm. The latest start time of task t_i is the latest time at which a task t_i may start in a way that t_i and all of its successors have the

chance to finish without violating any of their deadlines. The earliest start time is the longest path from an entry task to task t_i (Oh & Wu, 2004).

Definition 3.8. A task t_i will not miss the deadline if its execution starts (ST_i) in the time duration between latest start time (LST_i) and earliest start time (EST_i) of the task t_i . The task should start on or before the specified LST_i and on or after the specified EST_i of the task t_i .

$$EST_i \leq ST_i \leq LST_i \quad (3.4)$$

ST_i is the actual start of task t_i , EST_i is the earliest start time of task t_i and LST_i is the latest start time of task t_i on a specific processor.

Definition 3.9. The actual finish time of task t_i is given by

$$FT_i = ST_i + \tau_i \quad (3.5)$$

where ST_i is the actual start time of task t_i .

3.2.2 Power Model

We assume a multiprocessor system that has m voltage-scale processors $P = \{p_1, p_2, \dots, p_m\}$. Each processor can operate on a number of discrete voltage levels. The voltage level of a processor can dynamically be changed to one of its operating voltage. That is, each processor can switch its operating voltage from one level to another instantaneously and independently.

Energy efficient scheduling techniques have shown to minimize the run time energy consumption of a system. The dynamic power

consumption for CMOS circuits depends on the operating voltage and frequency of the processor (Jejurikar, 2005) and is given by:

$$pow_i = c_{eff} \cdot v_i^2 \cdot f_i \quad (3.6)$$

The dynamic power consumption to execute task t_i is denoted by pow_i , c_{eff} is the effective switching capacitance, v_i is the supply voltage assigned to execute task t_i and f_i is the operational frequency (frequency at run time).

The relation between power and voltage is quadratic, thus the decrease in the supply voltage decreases the power consumption of the processor. Scaling both supply and clock speed will produce at least quadratic power savings and a result quadratic energy savings (Falsafi & Vijaykumar, 2003). The technique of changing (scaling) the operating voltage and the clock frequency of processors dynamically is termed as dynamic voltage scaling DVS (Jejurikar, 2005).

The relationship between the supply voltage and the frequency is represented by the following equation:

$$f_i = \xi * \frac{(v_i - v_t)^2}{v_i} \quad (3.7)$$

Where ξ is the circuit dependent constant and v_t is the threshold voltage which is the lowest voltage supply, $v_i \gg v_t$ generally. The relation between f_i and v_i is nearly linear as given in the equation above. Hence the reduction in system speed will result in linear increase in time cost (Miao et al., 2008).

We assume that voltage and frequency are not continuous but there are l_k discrete levels of voltage and frequency pairs. System energy consumption can be reduced by controlling the operating voltage of system processors. Hence the operating voltage/frequency is assumed to be able to be scaled discretely at $k \in l_k$ levels.

Energy consumed by processing a single task t_i at voltage v_i and frequency f_i denoted as e_i is given by (Malakooti et al., 2013):

$$e_i = pow_i * \tau_i \quad (3.8)$$

Referring to (3.1), (3.6) and (3.8) then energy consumed by processing task t_i at voltage v_i and frequency f_i , denoted by e_i , is given by:

$$e_i = c_{eff} * v_i^2 * c_i \quad (3.9)$$

Where c_i is the number of cycles required to execute the task t_i .

Total energy, E , consumed to process all the n tasks is calculated by:

$$E = \sum_{i=1}^n e_i \quad (3.10)$$

e_{ijk} denotes energy consumed by task t_i when executed on processor p_j at voltage level VL_k and x_{ijk} is a decision variable defined as:

$$x_{ijk} = \begin{cases} 1, & \text{If task } i \text{ is allocated to procesor } j \text{ at voltage level } VL_k \\ 0, & \text{Otherwise} \end{cases} \quad (3.11)$$

The following example provides a numerical explanation of both tasks and power models.

Example 3.1: Assume having a job consisting of 3 real time tasks $T = \{t_1, t_2, t_3\}$. A periodic task is defined by $t_i = (c_i, d_i, FT_i, \tau_i)$, c_i is the workload required for execution of task t_i , d_i is the deadline, FT_i is the finishing time and τ_i the execution time of task t_i . Tasks may have precedence relation between them. Assume the following task set $T = \{t_1=(3,2,0.5,0.5), t_2=(4,5,0.75,0.75), t_3=(7,2,3,2.25)\}$. Corresponding values of execution time and finishing time are calculated using formulas in (3.1) and (3.5) respectively. Tasks are to be allocated on homogeneous multiprocessor system consisting of 3 processors $P = \{p_1, p_2, p_3\}$, in which each processor operates on 3 voltage levels. $VP_{p_j} = \{VL_1, VL_2, VL_3\}$. Where $j \in m$, $VL_k = (v_j, f_j)$, where $1 \leq k \leq lk$ and v_j is the voltage supply of processor p_j and f_j is the frequency at which a processor p_j operates at voltage level VL_k . Hence, $VL_1=(2, 1.5)$, $VL_2=(3, 2.7)$, $VL_3=(4, 3.75)$.

Considering the following possible solution illustrated in Figure (3.2):

Task	1	2	3
Processor	p ₁	p ₂	p ₃
Voltage level	VL ₁	VL ₂	VL ₃

Figure (3.2)

Power model example

Using the formulae (3.9) energy consumed for allocating a task t_i is on processor p_j and executed on voltage level VL_k is denoted as e_{ijk} . Hence;

$$e_{111} = 12, e_{222} = 36, e_{333} = 32$$

Then Using DVS the total energy consumed, E , is calculated using formulae (3.10)

$$E = 12 + 36 + 32 = 80$$

If we do not consider the use of DVS the total energy consumed when executing tasks on the highest voltage level of a processor then $E = 224$. The difference in saving energy then is 144.

3.3 Objective Function and Constraints

The main objective of the problem is minimizing Energy consumption of scheduling real-time tasks on multi-processor system. That is:

$$\text{Minimize } \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^{l_k} e_{ijk} * x_{ijk} \quad (3.12)$$

Subject to

$$ST_i \geq EST_i \quad \forall i \in [1, n] \quad (3.13)$$

$$FT_i \leq d_i \quad \forall i \in [1, n] \quad (3.14)$$

$$\sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^{l_k} x_{ijk} = 1 \quad \forall i \in [1, n] \quad (3.15)$$

$$VL_k \in VL_{p_j} \quad \forall k \in [1, l_k] \quad (3.16)$$

Constraint given in (3.13) states that the start execution time, ST_i of a task t_i should start on or after a specified earliest start time (ST_i should not start before EST_i). Constraint given in (3.14) specifies that the deadline of a specific task t_i should not surpass its finish time. In (3.15) the constraint states that each task t_i is assigned to a single processor p_j at voltage level VL_k . In constraints (3.16) VL_k is the voltage level which belongs to the set of voltage levels VL_{p_j} at which a processor j can operate on.

3.4 Summary

A clear definition of the problem is presented in the chapter, stating all kind of variable parameters affecting the problem and defining all constraints mathematically with a clear description of the notation used all through the derived formulation. The objective function described is subject to some constraints controlling its success if it was correctly satisfied.

Chapter Four

Proposed Algorithms

4.1 Genetic Algorithm	58
4.1.1 Solution Encoding	59
4.1.2 Generation of Initial Population	60
4.1.3 Crossover Operator	60
4.1.4 Mutation Operator	63
4.1.5 Replacement	66
4.1.6 Solution Validation	67
4.2 Cuckoo Search	68
4.2.1 Solution Encoding and Initial Solution Generation	70
4.2.2 Generating a Cuckoo Egg by Lévy Flight	70
4.2.3 Replace Worst Nests	72
4.2.4 Solution Validation	73
4.2.5 Dealing With Infeasible Solutions	73
4.3 Summary	73

Chapter Four

Proposed Algorithms

This chapter describes the main characteristics of the adapted algorithms. I present how basic parameters of both GA and cuckoo search were adapted to address various problem-specific issues.

4.1 Genetic Algorithm

Genetic algorithm (GA) is one of the most powerful optimization techniques for solving optimization problems (Paulinas & Ušinskas, 2007). With regard to task allocation problem, the GA was adapted accordingly. The pseudo-code of the adapted GA is given in Figure (4.1), with having re-designed parameters highlighted. The solution encoding and GA operators, namely, selection, crossover, mutation and replacement, are discussed with examples in the following sub-sections.

```
Generate initial population using algorithm given in Figure
(4.5)
Evaluate fitness of the population
Find the best chromosome
While (termination criteria not met)
{
    Selection(); //Select the parents for crossover
    Crossover(); //Perform crossover on the selected parents
    Mutation(); //Perform mutation
    For each child chromosome
        If (!Validate()) //Check feasibilities of the offspring
            Discard the chromosome
    Find the best chromosome so far
    Replacement(); //Select the population for next iteration
}
Return the best chromosome as solution
```

Figure (4.1)

Pseudo code of the adapted GA

4.1.1 Solution Encoding

In GA, individuals in the population are encoded into chromosomes. A chromosome represents a possible solution to the problem. Within the context of task allocation problem, a chromosome represents mapping of tasks to processors and voltage levels. Each chromosome is represented by 2-dimensional array as shown in Figure (4.2). Note that the array index indicates the task number; therefore, there is no need to store the task number in the array itself.

Task	1	2	3	4
Processor	P ₁	P ₂	P ₃	P ₂
Voltage level	V ₁	V ₂	V ₁	V ₃

Figure (4.2)
Solution encoding

Example1: Assume a system has 3 homogeneous processors; each operates at 3 different voltage levels v_1 , v_2 , and v_3 . There are 4 real-time tasks, as shown in Figure (4.3) as a DAG, to be allocated to the processor at suitable voltage levels.

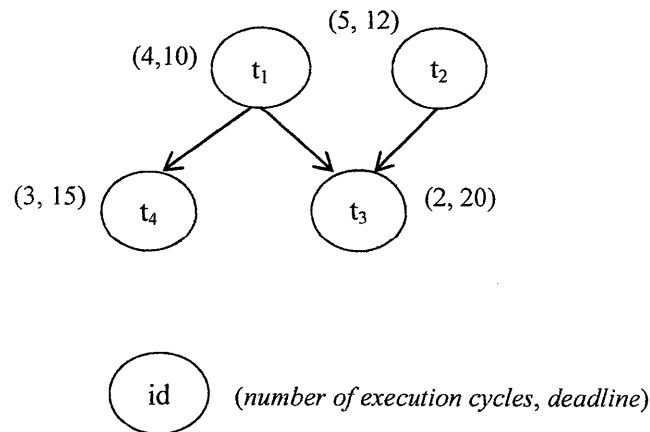


Figure (4.3)
A directed acyclic graph with four tasks

Figure 4.4 shows a valid chromosome in the solution set:

Tasks	t_1	t_2	t_3	t_4
Processor	1	2	3	2
Voltage level	2	1	3	3

Figure (4.4)
A Chromosome

4.1.2 Generation of Initial Population

GA starts with a set of random feasible set of solutions, called initial population. The algorithm given in Figure (4.5) was designed to generate the initial population. The algorithm creates each chromosome by selecting a random processor at which each task is allocated from the processors' set and a random voltage level at which the allocated task is executed.

```

s=1    /Counter for number of chromosomes
While (s<= PopSize) {
    for (i=1; i<=n; i++) {
        chromosome[1][i]=random(p)    //p ∈ {p1, p2, p3, ..., pm}
        chromosome[2][i]=random(v)    //v ∈ {v1, v2, v3, ..., vlk},
        // a voltage level the selected processor
    }
    If (Validate(chromosome)) {
        Evaluate(chromosome)
        Add chromosome in the population
        s++;
    }
    else discard the chromosome
}

```

Figure (4.5)
Generation of the initial population

4.1.3 Crossover Operator

To generate the offspring, a pair of chromosomes is selected from current population depending on the value of fitness function using the

roulette wheel approach. The selected parents are then subjected to the crossover operation. The crossover used in this work is a single point crossover. The crossover point is randomly chosen to divide each of the two selected parent chromosomes into two parts. The offspring is then generated by swapping the corresponding segments of the parents. I proposed three types of crossover as discussed below.

Type1 Crossover

Type1 crossover works by generating a single point crossover and then interchanged the corresponding values of the voltage level in the selected parents after the crossover point as illustrated in Figure (4.6). In the figure, the crossover point for parents 1 and 2 are marked with a vertical line. The crossover swaps the voltage level of parent 1 with the voltage level at the corresponding position in parent 2. This results in two offspring, namely, child 1 and child 2 as shown in the figure. Note that only voltage levels are interchanged according to their positions, without swapping the processor numbers.

Parent 1					Parent 2				
	t ₁	t ₂	t ₃	t ₄		t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P ₃	P ₂	processor	P ₁	P ₃	P ₁	P ₂
Voltage level	V ₁	V ₂	V ₁	V ₃	Voltage level	V ₃	V ₁	V ₃	V ₂

Child 1					Child2				
	t ₁	t ₂	t ₃	t ₄		t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P ₃	P ₂	processor	P ₁	P ₃	P ₁	P ₂
Voltage level	V ₁	V ₂	V ₃	V ₂	Voltage level	V ₃	V ₁	V ₁	V ₂

Figure (4.6)
Type1 crossover example

Type2 Crossover

Type2 crossover approach works by generating a single point crossover and then interchanging the corresponding values of the processors in the selected parents after the crossover point as illustrated in Figure (4.7). In the figure, the crossover point for parents 1 and 2 are marked with a vertical line. The crossover swaps the processor level of parent 1 with the processors at the corresponding position in parent 2. This results in two offspring, namely, child 1 and child 2 as shown in the figure.

Parent 1					Parent 2				
	t ₁	t ₂	t ₃	t ₄		t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P ₃	P ₂	processor	P ₁	P ₃	P ₁	P ₂
Voltage level	V ₁	V ₂	V ₁	V ₃	Voltage level	V ₃	V ₁	V ₃	V ₂

Child 1					Child2				
	t ₁	t ₂	t ₃	t ₄		t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P ₁	P ₂	processor	P ₁	P ₃	P ₃	P ₂
Voltage level	V ₁	V ₂	V ₁	V ₃	Voltage level	V ₃	V ₁	V ₃	V ₂

Figure (4.7)
Type2 crossover example

Type3 Crossover

Type 3 crossover works by generating a single point crossover and then interchanging the corresponding values of the voltage level and processors in the selected parents after the crossover point as illustrated in Figure (4.8).

In the figure, the crossover point for parents 1 and 2 are marked with a vertical line. The crossover swaps the voltage level and processors of parent 1 with the processor level at the corresponding position in parent 2. This results in two offspring, namely, child 1 and

child 2 as shown in the figure. Note that both of voltage levels and processors are interchanged according to their positions.

Parent 1					Parent 2				
	t ₁	t ₂	t ₃	t ₄		t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P ₃	P ₂	processor	P ₁	P ₃	P ₁	P ₂
Voltage level	V ₁	V ₂	V ₁	V ₃	Voltage level	V ₃	V ₁	V ₃	V ₂

Child 1					Child 2				
	t ₁	t ₂	t ₃	t ₄		t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P ₁	P ₂	processor	P ₁	P ₃	P ₃	P ₂
Voltage level	V ₁	V ₂	V ₃	V ₂	Voltage level	V ₃	V ₁	V ₁	V ₃

Figure (4.8)
Type 3 crossover example

4.1.4 Mutation Operator

Mutation operator works by choosing a random gene of a selected chromosome and changing the gene to a new gene. Four types of mutations are proposed as explained below.

Type1 Mutation

In Type1 mutation, a randomly selected gene is mutated by changing the processor and voltage level assigned to a task. This mutation is done in a random manner; any voltage-processor pair is selected randomly and replaced with other values of voltage and processor. Figure (4.9) illustrates Type 1 mutation. In this figure, a chromosome that is selected for mutation is shown. In this chromosome, the processor-voltage pair, say (P₃,V₁) at the position of task t₃, is randomly selected. Then, the processor P₃ is replaced by processor P₂ (where processor P₂ is selected randomly). Similarly, voltage level V₁ is replaced by voltage level V₃ (again through a random selection). This results in the mutated chromosome as depicted in the figure.

Chromosome				
	t_1	t_2	t_3	t_4
Processor	P_1	P_2	P_3	P_2
Voltage level	V_1	V_2	V_1	V_3

Mutated chromosome				
	t_1	t_2	t_3	t_4
Processor	P_1	P_2	P_2	P_2
Voltage level	V_1	V_2	V_3	V_3

Figure (4.9)
Type1 mutation example

Type2 Mutation

This approach proceeds by selecting two random genes of the selected chromosome and then the processor-voltage pairs in these genes are swapped. For example, in Figure (4.10), two randomly selected genes are at positions of tasks t_1 and t_3 . The processor-voltage pair in these two genes are (P_1, V_1) and (P_3, V_1) respectively. When these pairs are swapped, the resultant is a mutated chromosome which is also depicted in Figure (4.10).

Chromosome				
	t_1	t_2	t_3	t_4
Processor	P_1	P_2	P_3	P_2
Voltage level	V_1	V_2	V_1	V_3

Mutated chromosome				
	t_1	t_2	t_3	t_4
Processor	P_3	P_2	P_1	P_2
Voltage level	V_1	V_2	V_1	V_3

Figure (4.10)
Type2 mutation example

Type3 Mutation

In this type, the chromosome is mutated by selecting the gene randomly and changing the voltage level to another randomly selected voltage level. This process is illustrated through an example in the Figure (4.11) where the position of task t_3 is randomly selected. The voltage level at t_3 is V_1 which is replaced by another randomly selected voltage level, say V_2 , as depicted in the figure.

Chromosome				
	t_1	t_2	t_3	t_4
Processor	P_1	P_2	P_3	P_2
Voltage level	V_1	V_2	V_1	V_3

Mutated chromosome				
	t_1	t_2	t_3	t_4
Processor	P_1	P_2	P_3	P_2
Voltage level	V_1	V_2	V_2	V_3

Figure (4.11)
Type3 mutation example

Type4 Mutation

In this approach, a gene is randomly selected and mutation is done only at the processor level. The processor in the selected gene is replaced with another randomly selected processor. Figure (4.12) shows an example as how it is done. In this figure, processor P_3 at task t_3 is randomly selected for mutation. This processor is then randomly replaced with processor P_2 at the same position as depicted in the figure.

Chromosome				
	t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P ₃	P ₂
Voltage level	V ₁	V ₂	V ₁	V ₃

Mutated chromosome				
	t ₁	t ₂	t ₃	t ₄
Processor	P ₁	P ₂	P₂	P ₂
Voltage level	V ₁	V ₂	V ₁	V ₃

Figure (4.12)
Type4 mutation example

It is important to mention that in all types of mutations discussed above, the resultant mutated chromosomes are tested for validity and duplication. Based on the outcome of the test, it is decided accordingly whether to accept the new (mutated) chromosome or not. If the new chromosome is accepted then it is added to the newly generated population. The chromosome passes through the selection function to generate the final population that will be passed onto the GA algorithm in the next iteration.

4.1.5 Replacement

During successive generation, a part of the existing population is selected to breed a new generation. The selection of individual solutions is done through fitness based process. In this work, two techniques have been adopted for the replacement operator; selection of solutions using roulette wheel technique, and elitist selection.

Roulette Wheel Selection

Since I am using the roulette wheel selection to solve a minimization problem I developed it in a special way to guarantee that it works appropriately. Each solution in the population is assigned

probability of selection (between 0 and 1) in a way that better solutions that have higher fitness have higher probability to be selected each time.

Elitist Selection

This approach is implemented strictly on the level of fitness of the newly generated chromosomes. The population in the next iteration will include the children chromosome that exhibit the best level of fitness values. The best children and the best chromosomes of the original population are added to the new population in case that the children chromosomes are less than the total size of the population.

4.1.6 Solution Validation

To ensure that the population always contains valid and feasible solution, a validation check is performed as given in Figure (4.13). Constraints validation algorithm calculates and tests the values of EST_i , LST_i , ST_i , and FT_i for each task in a chromosome against the constraints. If the chromosome is valid, the algorithm returns true, otherwise it returns false.

```

Boolean Validate(chromosome) {
    Define FlowT[m]={0};           //m is number of processors
                                   //defines the availability
                                   //of each selected processor

    For(i=1; i<=n; i++){
        p=chromosome [1][i]       //p∈ {p1, p2, p, ..., pm}
        v=chromosome [2][i]       // v ∈ {v1, v2, v3, ..., vlk}
                                   //voltage level for the selected
                                   //processor

        f=ξ * (v-vt)2/ v         //ξ is the circuit dependent constant
                                   //vt is the threshold voltage which is
                                   //the lowest voltage supply

        τi = ci /f
        if(Pre(ti) = ϕ){
            LSTi= di - τi         //LSTi: latest start time of ti
            ESTi= 0                 //ESTi: earliest start time of ti
            STi= FlowT[p]           //STi start time of ti
            FTi= τi + STi
            FlowT[p]= FlowT[p]+τi   //update availability for the
                                   //processor
        }
        else {
            LSTi= di - τi
            ESTi= maxtj∈Pre(ti) (FTj) //Find maximum FTj among all
                                   //predecessors of task tj.

            STi= FlowT[p] + ESTi
            FTi= τi + STi         //FTi Finish time of ti
            FlowT[p]= FlowT[p]+τi   //update availability for the
                                   //processor
        }
    }
    For (i=1; i<=n; i++){
        if((STi ≤ LSTi)AND(STi ≥ ESTi)AND(FTi ≤ di)) continue;
        else return false;
    }
    return true;
}

```

Figure (4.13)
Constraints validation algorithm

4.2 Cuckoo Search

Cuckoo search (CS) described in Chapter 2 is adapted in this work to address the problem under study, details of which are given below. Figure (4.14) presents the main steps of the adapted cuckoo search algorithm. An initial population is generated using the algorithm that was

used to generate the initial population for genetic algorithm. During each iteration, a random solution (i.e. egg) is selected and modified by changing specific values of processor and voltage level. This is done using Lévy flight around the best solution obtained so far to speed up the local search. The resultant new solution will replace another randomly selected egg if the new solution is of better quality. A fraction of the population consisting of low quality solutions are replaced with new randomly generated solutions which makes the population for the next iteration.

The original CS algorithm was designed for problems in the continuous domain. However, this approach is not applicable to the underlying problem due to the problem being in discrete domain. Therefore, necessary adaptations are required which are highlighted in Figure (4.14). One major change is the discrete step size. Details of the adapted CS are discussed in the following subsections.

Generate initial population using algorithm given in Figure (4.5)

```

Find egg  $s_{best}$  with best fitness value
While (termination criterion not met) {
    Pick a random egg ( $s_r$ ) from the population
    step_size=Calculate_SZ( $s_r, s_{best}$ )
    Evaluate fitness  $F_r = f(s_r)$ 
    Generate  $s_i$ =GenerateSolution(step_size,  $s_r, s_{best}$ )
    Evaluate fitness  $F_i = f(s_i)$ 
    if (!Validate ( $s_i$ )) then discard the solution
    Choose an egg randomly ( $s_j$ )
    if ( $F_i < F_j$ )  $s_j = s_i$ 
    Sort nests on fitness values
    Replace a fraction ( $p_\alpha$ ) of eggs having worst fitness value with
    new random solutions
    Rank solutions
    Find best solution of the iteration ( $c_{best}$ )
    if ( $f(c_{best}) < f(s_{best})$ )  $s_{best} = c_{best}$ 
}
return  $s_{best}$  as a solution

```

Figure (4.14)
Adapted cuckoo search algorithm

4.2.1 Solution Encoding and Initial Solution Generation

Encoding and initial population generation techniques discussed in Sections 4.1.1 and 4.1.2 respectively are also used in the CS algorithm.

4.2.2 Generating a Cuckoo Egg by Lévy Flight

Since the original cuckoo search was developed for continuous domain, and the problem in this study is in the discrete domain, the changing process is adjusted accordingly. The step size which is related to the amount of changes should be taking place on the random egg s_r to generate the new egg s_i . The parameter λ defines the step size and α defines the count of changes to be performed to generate the egg s_i . The scale of this random search is controlled by multiplying the generated Lévy flight by α .

$$stepsize = Levy(\lambda) \times \alpha \quad (4.1)$$

Algorithm in Figure (4.15) presents the main steps of calculating the step size.

```
Calcutte_SZ( $s_r$ ,  $s_{best}$ )
{
    countDif=0;                                //stores number of possible changes
    for(i=1; i<=n; i++)
    {
        if ( $s_r[1][i] \neq s_{best}[1][i]$  or  $s_r[2][i] \neq s_{best}[2][i]$ ){
            change[countDif]=i;
            CountDif= countDif+1;
        }
        else continue;
    }
    stepsize = $\lambda$  * countDif;
    return stepsize;
}
```

Figure (4.15)
Calculate step size algorithm

Using the step size, a new egg is generated using the algorithm in Figure (4.16).

```

GenerateSolution(step_size, sr, sbest, countDif)
{
    While (step_size>0)
    {
        changeG= random(countDif); //n is total number of tasks
                                   //changeG is a variable to select
                                   //a random task to be changed
        i= change[changeG];
        si[1][i]=sbest[1][i];
        si[2][i]=sbest[2][i];
        step_size = step_size-1;
    }
}

```

Figure (4.16)
Generate new solution algorithm

The following is a numerical explanation of the algorithms of calculating step size and generating a new solution using Lévy flight as described in Figures (4.15 and 4.16).

Example2: Assume having the following values of λ and p_α

$$\lambda = 0.5$$

$$p_\alpha = 0.75$$

Considering the random solution (s_r) represented in Figure (4.17) we find that $\text{step_size} = 2$. Step_size is calculated using the algorithm in Figure (4.15). Step size indicates the number of changes required to generate a solution of better quality out of the random solution. the step size is determined using λ and the count of differences between the random solution and the best solution (α) as follows:

$$\text{step_size} = \text{approximate result of } (\lambda \times \alpha)$$

$$\text{Step size} = 0.5 \times 3 = 1.5 \cong 2$$

The approximation is either floor approximation or ceiling approximation of the resulting step size depending on the fraction decimal value. In the example we used ceiling approximation since the fraction decimal value ≥ 0.5 , otherwise floor approximation of the value is considered. New solution is generated using step_size as illustrated in figure (4.17 D). The newly generated solution is then checked for validity and repetition and then compared to a random solution (s_j). The random solution is replaced with the new generated solution if it is of better quality.

A. Overall best solution

t_1	t_2	t_3	t_4
P_2	P_1	P_3	P_2
V_1	V_3	V_2	V_2

B. Random selected solution s_r

t_1	t_2	t_3	t_4
P_3	P_2	P_1	P_2
V_3	V_3	V_1	V_2

C. Random solution s_j

t_1	t_2	t_3	t_4
P_2	P_1	P_2	P_3
V_1	V_1	V_2	V_2

D. The resultant modified solution s_i

t_1	t_2	t_3	t_4
P_2	P_1	P_1	P_2
V_1	V_3	V_1	V_2

Figure (4.17)

Example of generating new solution and stepsize calculation

4.2.3 Replace Worst Nests

In the proposed cuckoo search, a number of worst eggs are replaced with new randomly generated eggs. The number of solutions to be replaced is determined by p_a (the fraction of total population that should be replaced). For example if total number of nests=20 and $p_a=0.5$ then the worst 10 solutions are to be replaced by new random solutions.

4.2.4 Solution Validation

The constraints validation described in Section 4.1.6 to check feasibility of each solution is used in this algorithm as well.

4.2.5 Dealing with Infeasible Solutions

Two approaches have been adopted to deal with the infeasible solutions. In the first approach, infeasible solutions are discarded. In the second approach, the population can have a pre-specified percentage of infeasible solutions but with a penalty. The penalty is calculated by adding predefined percentage of the fitness value of the worst feasible solution to the fitness value of the infeasible solutions. For example, if $\mathbf{x}'_i$ is infeasible solution and $\mathbf{x}_j$ is the worst valid solution then the penalty is calculated by adding a percentage ($y\%$) of the fitness of the worst valid solution to the fitness of the invalid solution using the following formulae $f(\mathbf{x}'_i) = f(\mathbf{x}'_i) + y\% * f(\mathbf{x}_j)$. However this technique applies extra overhead per iteration since it keeps the count of those solutions which are added with penalties.

4.3 Summary

In this chapter we presented the adapted genetic algorithm and cuckoo search. We also discussed in detail specific algorithmic operators.

Chapter Five

Results and Discussion

5.1 Characteristics of Test Cases	75
5.2 Experimental procedure	76
5.3 Results and discussion of Genetic Algorithm	76
5.3.1 Parameter Settings	77
5.3.2 Effect of Parameter Values	79
5.3.2.1 Effect of Population Size	79
5.3.2.2 Effects of Crossover and Mutation Operators	82
5.3.2.3 Effect of Replacement Operator	84
5.3.3 Experimentation with Selected Parameter Combinations	85
5.4 Results and Discussion of Cuckoo Search	87
5.4.1 Parameter Settings	88
5.4.2 Effect of Parameter Values	88
5.4.2.1 Effect of Number of Nests	89
5.4.2.2 Effect of λ and p_a Values	91
5.4.3 Experimentation with Selected Parameter Combinations	92
5.5 Results and Discussion of Cuckoo Search with Penalty	94
5.5.1 Parameter Setting	95
5.5.2 Effect of parameter Values	95
5.5.2.1 Effect of Number of Nests	95
5.5.2.2 Effect of λ and p_a Values	97
5.5.3 Experimentation with Selected Parameter Combinations	98
5.6 Comparison of Algorithms	100
5.6.1 Solution Quality Analysis	100
5.6.2 Comparison of Run Time	102
5.6.3 Percentage Saving in Power	104
5.6.4 Statistical Test Results	106
5.7 Effect of Changing the Number of Processors on Algorithm Performance	107
5.8 Summary	109

Chapter Five

Results and Discussion

The previous chapter provided a description of the adapted algorithms. This chapter presents a discussion on the experiments conducted and their results to evaluate the performance of the proposed algorithms. The performance is assessed based on the quality of solutions produced and the time efficiency. First, the empirical study related to genetic algorithm is presented. The effect of different algorithm parameters and operators is investigated. Later, the experiments and results related to cuckoo search are discussed. Investigation of the different algorithm parameters for cuckoo search is also provided. Finally, a comparison of the adapted algorithms is provided with respect to solution quality and time efficiency.

5.1 Characteristics of Test Cases

Six directed acyclic graphs (DAGs) were used for the experimental study. TGFF (Dick, Rhode & Wolf, 1998) was used to generate task graphs for the experiments. Each of these graphs represents the set of real-time tasks and related information (execution time cycle, deadline, dependencies). Number of processors was taken as 5, 7, and 9. Each processor operated at 3 levels of voltage. The characteristics of test cases used in the study are summarized in Table (5.1).

Table (5.1)
Characteristics of Test cases used in experiments

Test case	Number of tasks
1	10
2	15
3	30
4	50
5	75
6	100

5.2 Experimental Procedure

All simulations were run on i7 processors with 8 GB RAM and coding was done in C++. With regard to all algorithms, 30 independent runs were conducted for each parameter setup, and the average of best solutions found in each run was recorded along with the standard deviation. The average run time over the 30 runs for each test case was also recorded. Details of these parameters are given in the following sections. The purpose of these experiments was to study the effect of various algorithmic parameters on the quality of solution and execution time. All results were also validated through statistical testing with Wilcoxon-ranked-sum test using Statistica 5.1 package. A confidence level of 95 % was used. It is important to mention that both algorithms were given the same seed solution.

5.3 Results and Discussion of Genetic Algorithm

This section presents the conducted experiments along with the discussion of the empirical results. A detailed study of the operators and algorithm parameters is provided.

5.3.1 Parameter Settings

The parameter settings and their combinations (a total of 96 combinations) used in the study are given in Tables (5.2) and (5.3) respectively. The values for the population size and iterations were motivated by the typical values reported in relevant literature (Pencheva et al., 2015; Roeva et al., 2013; Alajmi & Wright, 2014). Sensitivity analysis was used to tune the parameters using different DAGs with 10 and 15 tasks. The best combinations then were used on the remaining test cases.

Table (5.2)
Parameter settings for GA

Population size	20, 40, 60
Iterations	100, 200, 300
Mutation rate	0.05, 0.1
Crossover rate	0.6, 0.8

Table (5.3)
GA combination testing set

Combination	Parameters				
	Crossover*		Mutation**		Replacement Type ⁺
	Type	Rate (P _c)	Type	Rate (P _m)	
C1	Type 3	0.6	Type 2	0.05	Type 2
C2	Type 3	0.6	Type 2	0.1	Type 2
C3	Type 3	0.8	Type 2	0.05	Type 2
C4	Type 3	0.8	Type 2	0.1	Type 2
C5	Type 3	0.6	Type 2	0.1	Type 1
C6	Type 3	0.6	Type 2	0.05	Type 1
C7	Type 3	0.8	Type 2	0.05	Type 1
C8	Type 3	0.8	Type 2	0.1	Type 1
C9	Type 1	0.6	Type 2	0.05	Type 2
C10	Type 1	0.6	Type 2	0.1	Type 2
C11	Type 1	0.8	Type 2	0.05	Type 2
C12	Type 1	0.8	Type 2	0.1	Type 2
C13	Type 1	0.6	Type 2	0.05	Type 1
C14	Type 1	0.6	Type 2	0.1	Type 1
C15	Type 1	0.8	Type 2	0.05	Type 1
C16	Type 1	0.8	Type 2	0.1	Type 1

C17	Type 2	0.6	Type 2	0.05	Type 2
C18	Type 2	0.6	Type 2	0.1	Type 2
C19	Type 2	0.8	Type 2	0.05	Type 2
C20	Type 2	0.8	Type 2	0.1	Type 2
C21	Type 2	0.6	Type 2	0.05	Type 1
C22	Type 2	0.6	Type 2	0.1	Type 1
C23	Type 2	0.8	Type 2	0.05	Type 1
C24	Type 2	0.8	Type 2	0.1	Type 1
C25	Type 3	0.6	Type 1	0.05	Type 2
C26	Type 3	0.6	Type 1	0.1	Type 2
C27	Type 3	0.8	Type 1	0.05	Type 2
C28	Type 3	0.8	Type 1	0.1	Type 2
C29	Type 3	0.6	Type 1	0.05	Type 1
C30	Type 3	0.6	Type 1	0.1	Type 1
C31	Type 3	0.8	Type 1	0.05	Type 1
C32	Type 3	0.8	Type 1	0.1	Type 1
C33	Type 1	0.6	Type 1	0.05	Type 2
C34	Type 1	0.6	Type 1	0.1	Type 2
C35	Type 1	0.8	Type 1	0.05	Type 2
C36	Type 1	0.8	Type 1	0.1	Type 2
C37	Type 1	0.6	Type 1	0.05	Type 1
C38	Type 1	0.6	Type 1	0.1	Type 1
C39	Type 1	0.8	Type 1	0.05	Type 1
C40	Type 1	0.8	Type 1	0.1	Type 1
C41	Type 2	0.6	Type 1	0.05	Type 2
C42	Type 2	0.6	Type 1	0.1	Type 2
C43	Type 2	0.8	Type 1	0.05	Type 2
C44	Type 2	0.8	Type 1	0.1	Type 2
C45	Type 2	0.6	Type 1	0.05	Type 1
C46	Type 2	0.6	Type 1	0.1	Type 1
C47	Type 2	0.8	Type 1	0.05	Type 1
C48	Type 2	0.8	Type 1	0.1	Type 1
C49	Type 3	0.6	Type 3	0.05	Type 2
C50	Type 3	0.6	Type 3	0.1	Type 2
C51	Type 3	0.8	Type 3	0.05	Type 2
C52	Type 3	0.8	Type 3	0.1	Type 2
C53	Type 3	0.6	Type 3	0.05	Type 1
C54	Type 3	0.6	Type 3	0.1	Type 1
C55	Type 3	0.8	Type 3	0.05	Type 1
C56	Type 3	0.8	Type 3	0.1	Type 1
C57	Type 1	0.6	Type 3	0.05	Type 2
C58	Type 1	0.6	Type 3	0.1	Type 2
C59	Type 1	0.8	Type 3	0.05	Type 2
C60	Type 1	0.8	Type 3	0.1	Type 2
C61	Type 1	0.6	Type 3	0.05	Type 1
C62	Type 1	0.6	Type 3	0.1	Type 1
C63	Type 1	0.8	Type 3	0.05	Type 1
C64	Type 1	0.8	Type 3	0.1	Type 1
C65	Type 2	0.6	Type 3	0.05	Type 2
C66	Type 2	0.6	Type 3	0.1	Type 2
C67	Type 2	0.8	Type 3	0.05	Type 2
C68	Type 2	0.8	Type 3	0.1	Type 2
C69	Type 2	0.6	Type 3	0.05	Type 1
C70	Type 2	0.6	Type 3	0.1	Type 1
C71	Type 2	0.8	Type 3	0.05	Type 1
C72	Type 2	0.8	Type 3	0.1	Type 1
C73	Type 3	0.6	Type 4	0.05	Type 2

C74	Type 3	0.6	Type 4	0.1	Type 2
C75	Type 3	0.8	Type 4	0.05	Type 2
C76	Type 3	0.8	Type 4	0.1	Type 2
C77	Type 3	0.6	Type 4	0.05	Type 1
C78	Type 3	0.6	Type 4	0.1	Type 1
C79	Type 3	0.8	Type 4	0.05	Type 1
C80	Type 3	0.8	Type 4	0.1	Type 1
C81	Type 1	0.6	Type 4	0.05	Type 2
C82	Type 1	0.6	Type 4	0.1	Type 2
C83	Type 1	0.8	Type 4	0.05	Type 2
C84	Type 1	0.8	Type 4	0.1	Type 2
C85	Type 1	0.6	Type 4	0.05	Type 1
C86	Type 1	0.6	Type 4	0.1	Type 1
C87	Type 1	0.8	Type 4	0.05	Type 1
C88	Type 1	0.8	Type 4	0.1	Type 1
C89	Type 2	0.6	Type 4	0.05	Type 2
C90	Type 2	0.6	Type 4	0.1	Type 2
C91	Type 2	0.8	Type 4	0.05	Type 2
C92	Type 2	0.8	Type 4	0.1	Type 2
C93	Type 2	0.6	Type 4	0.05	Type 1
C94	Type 2	0.6	Type 4	0.1	Type 1
C95	Type 2	0.8	Type 4	0.05	Type 1
C96	Type 2	0.8	Type 4	0.1	Type 1

*Crossover type: Type 1: Voltage crossover, Type 2: Processor crossover, Type 3: Voltage and processor crossover.

**Mutation Type: Type 1: Random mutation, Type 2: Swap mutation, Type 3: Voltage mutation, Type 4: Processor mutation.

* Replacement Type: Type 1: Roulette wheel, Type 2: Elitist selection

5.3.2 Effect of Parameter Values

The effect of different GA parameters on the quality of solution and run time is discussed below. These include the effects of population size, crossover, and mutation types and rates. The test cases considered for sensitivity analysis are test case1 (# of tasks 10 tasks) and test case2 (# of tasks 15 tasks) on a multi-processor system composed of 5 processors.

5.3.2.1 Effect of Population Size

As mentioned by Pencheva et al. (2015) that the number of chromosomes in the population has a great effect on the overall performance of the algorithm. Fewer chromosomes will provide a few possibilities to perform crossover and only a small part of search space to

be explored. On the other hand, larger number of chromosomes will slow down GA. It is proposed by Pencheva et al. (2015) that a population size of 20-30 chromosomes is good, though a size of 50-100 chromosomes are reported to be best by Obitko (2005).

The quality of solution for population size of 20, 40 and 60 chromosomes was tested. As can be observed from the results shown in Figure (5.1) to Figure (5.4), the population size has an effect on the solution quality. The results show that the GA was able to find better quality solutions with population size 40 as compared to population size of 20. Increasing the population size further to 60 did not improve the quality of solutions any further. Moreover, the average run time analysis as shown in Figure (5.5) and Figure (5.6) indicating that GA converges faster with a population size of 40 as compared to population size of 60. Therefore, a population size of 40 found to be suitable for further experimentation for other test cases.

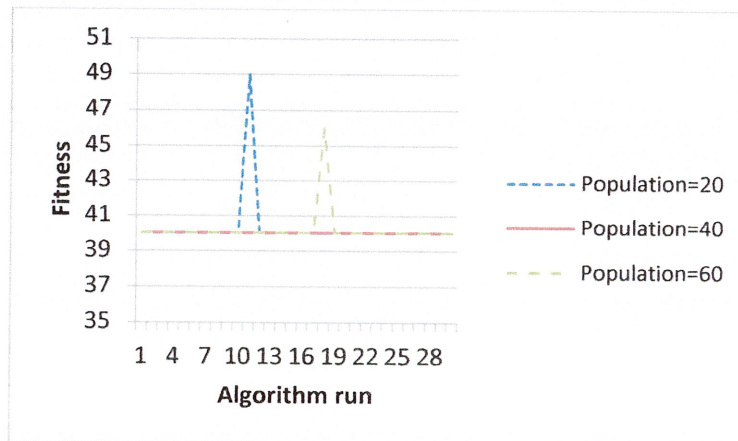


Figure (5.1)

Fitness values obtained during the 30 runs for 20, 40, and 60 chromosomes in the population for Test case 1 for the combination: Type 3 crossover (rate= 0.8), Type 1 mutation (rate= 0.1) and Type 2 replacement

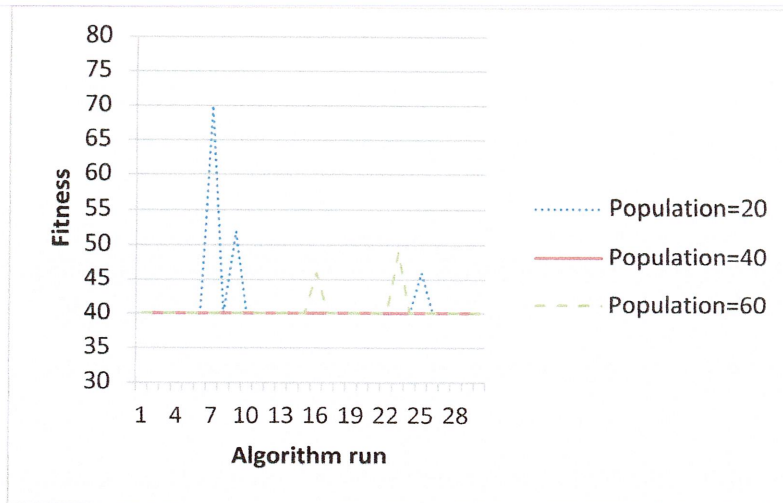


Figure (5.2)

Fitness function values obtained during the 30 runs for 20, 40, and 60 chromosomes in the population for Test case1 for the combination: Type 3 crossover (rate= 0.6), Type 3 mutation (rate= 0.05) and Type 2 replacement

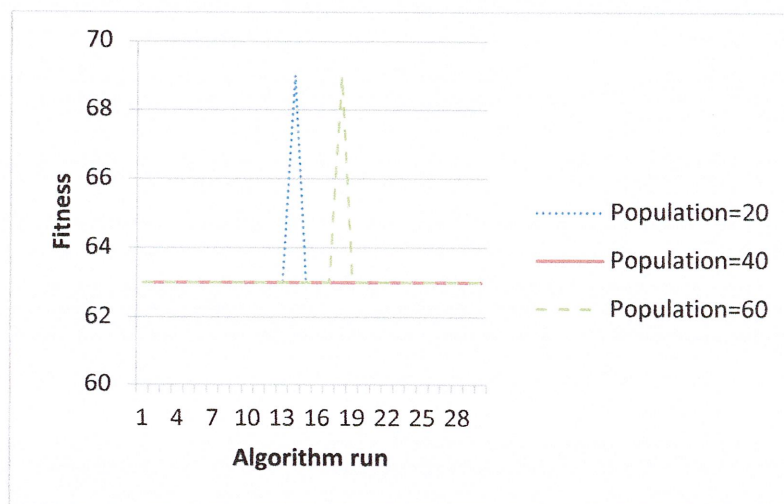


Figure (5.3)

Fitness function values obtained during the 30 runs for 20, 40, and 60 chromosomes in the population for Test case2 for the combination: Type 3 crossover (rate=0.8), Type 1 mutation (rate= 0.05) and Type 1 replacement

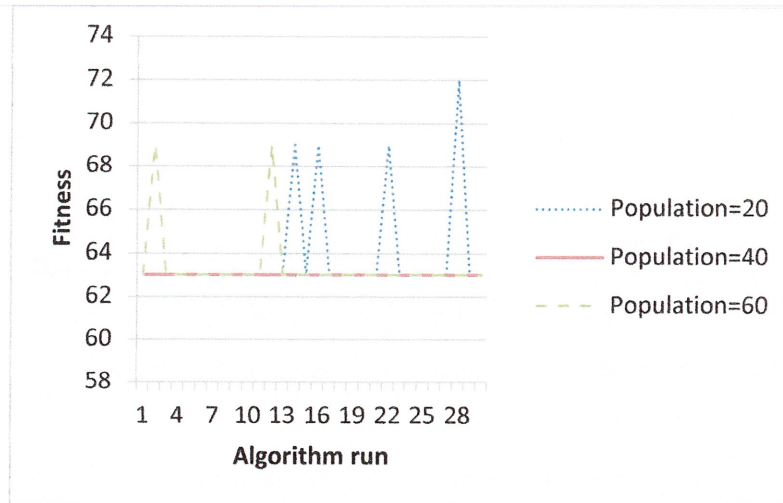


Figure (5.4)

Fitness function values obtained during the 30 runs for 20, 40, and 60 chromosomes in the population for Test case2 for the combination: Type 2 crossover (rate=0.8), Type 1 mutation (rate= 0.05) and Type 1 replacement

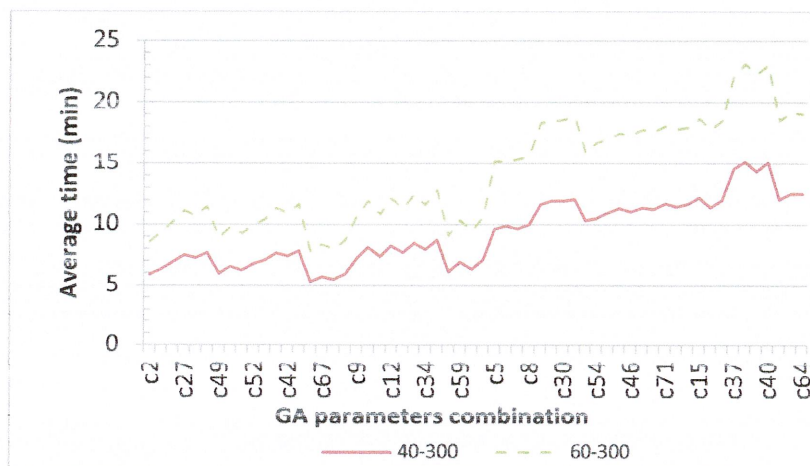


Figure (5.5)

Effect of population size on average execution time on Test case1



Figure (5.6)

Effect of population size on average execution time on Test case2

5.3.2.2 Effects of Crossover and Mutation Operators

The main controlling operators of GA are crossover and mutation. The performance of the algorithm is governed by crossover and mutation rates. Figure (5.7) and Figure (5.8) show that many combinations are not producing higher quality results and therefore were not considered for further experimentation.

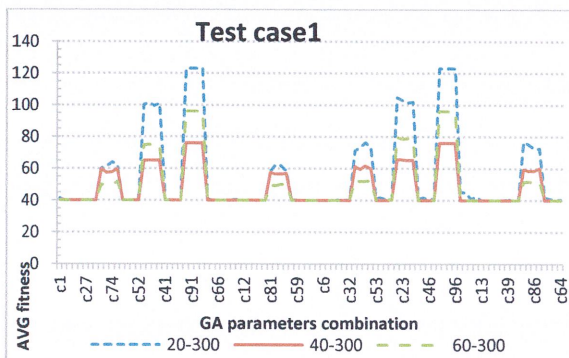


Figure (5. 7)

Effect of GA combinations on solution quality of Test case1 (300 iteration, 20, 40, and 60 chromosomes)

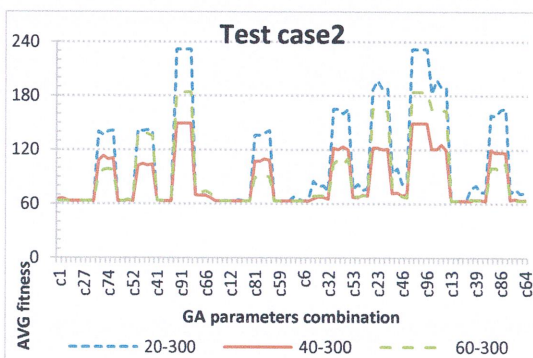


Figure (5. 8)

Effect of GA combinations on solution quality of Test case2 (300 iteration, 20, 40, and 60 chromosomes)

The selected combinations of crossover and mutation were further subjected to short listing by considering execution time. Figure (5.5) and Figure (5.6) show that combinations C2, C5, C7, C10 and C14 generated results with less execution time. Therefore, since these combinations resulted in high quality solutions and less execution time, they were used for further experimentation with other test cases. To be more specific the number of combinations was minimized to 5 compared to 96 combinations originally considered.

5.3.2.3 Effect of Replacement Operator

The effect of the replacement parameter was studied while keeping the population size of 40 and with the best combinations of crossover and mutation identified earlier. The testing was done on two types of replacement, namely, roulette wheel and elitist approach. Results are depicted in Tables (5.4) and (5.5), which indicate that the following combinations produced the best results.

Table (5. 4)
Average fitness of best combinations for Test case2

Population size	C2	C4	C5	C6	C7	C14	C16	C5	C10
	AvgFitness	AvgFitness	AvgFitness	AvgFitness	AvgFitness	AvgFitness	AvgFitness	AvgFitness	AvgFitness
40	63	63	63	63	63	63	63	63	63
60	63	63	63	63	63	63	63	63	64

Table (5. 5)
Average execution time of best combinations for Test case 2

Population size	C2 Avg Time(min)	44C Avg Time(min)	85C Avg Time(min)	060 Avg Time(min)	7C Avg Time(min)	C14 Avg Time(min)	91C Avg Time(min)	C5 Avg Time(min)	C10 Avg Time(min)
40	7	13.5	11	12	15	14	15	15	8
60	11	14.3	12	14	21	22	23	21	12
MIN	7	13.5	11	12	15	14	15	15	8

Eventually, the following combinations were used for further experimentation with other test cases.

- Type3 crossover ($P_c = 0.6$), Type 2 Mutation ($P_m = 0.1$), Replacement with Elitist Selection
- Type 3 crossover($P_c = 0.6$), Type 2 Mutation ($P_m = 0.1$), Replacement with RW Selection
- Type 1 Crossover ($P_c = 0.6$), Type 2 Mutation ($P_m = 0.1$), Replacement with Elitist Selection
- Type 1 Crossover ($P_c = 0.6$), Type 2 Mutation ($P_m = 0.1$), Replacement with RW Selection
- Type 3 Crossover ($P_c = 0.8$), Type 2 Mutation($P_m=0.05$), Replacement with RW Selection

5.3.3 Experimentation with Selected Parameter Combinations

The second step after defining the set of the best combinations as discussed above is studying the effect of those combinations on other test cases.

Figure (5.9) depict the effect of the 5 selected combinations on the runtime for all remaining test cases while considering multiprocessor system consisting of 5, 7, and 9 processors. It is observed from these figures that for most cases, combinations c2 and c10 were the best.

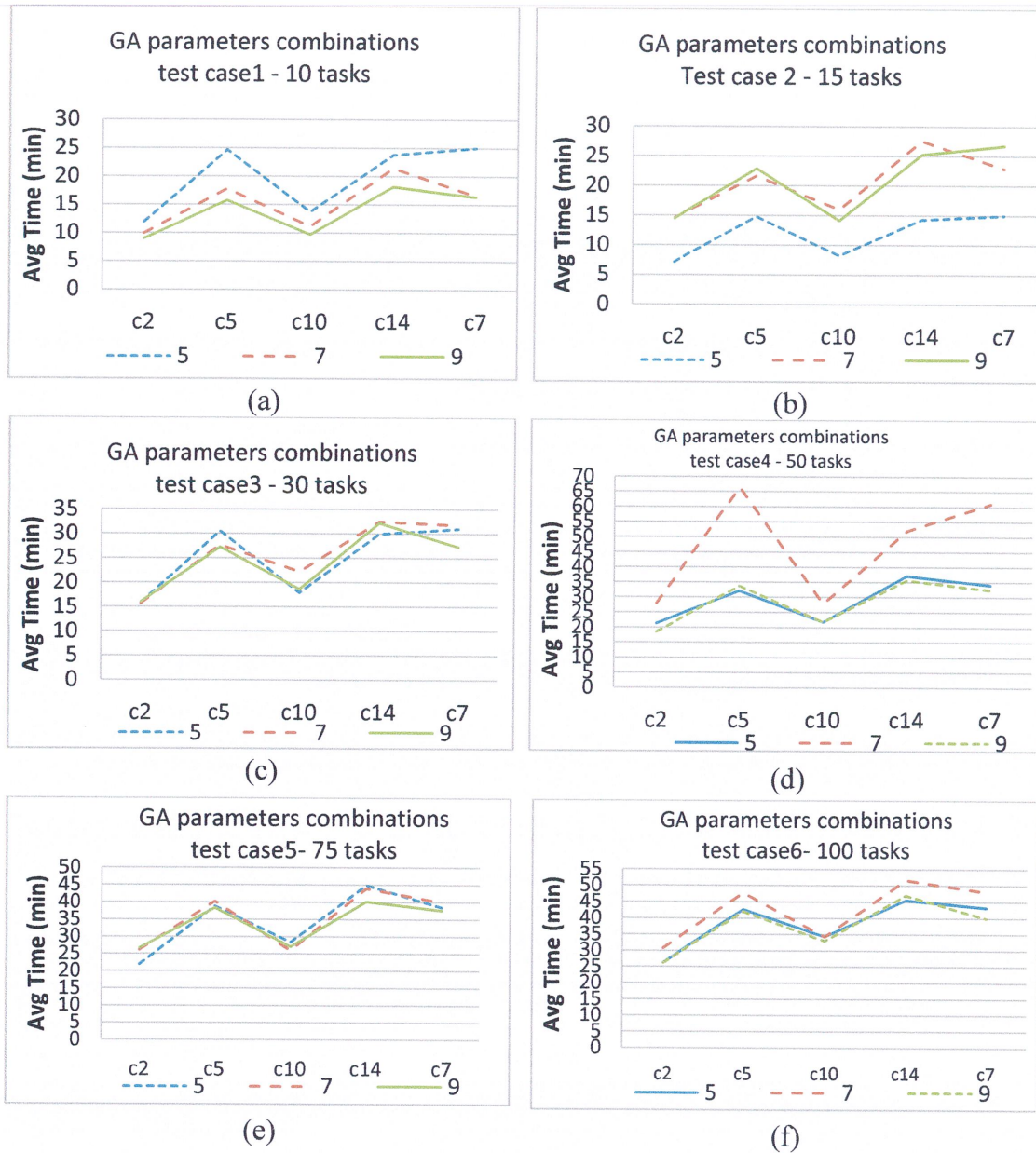


Figure (5.9)

Time analysis of all test cases with the best combinations on different multiprocessor systems

With regard to the quality of solutions, Table (5.6) shows the best results obtained with the selected combinations of operators. The combinations have been tested on all test cases on different multiprocessor systems composed of 5, 7 and 9 processors. It is observed from the table that combination C10, which represents Type 1 crossover, Type 2 mutation (corresponding to $P_c = 0.6$ and $P_m = 0.1$ respectively), and elitist

replacement showed the best performance, although the difference in performance with different number of processors was not very different from each other.

Table (5. 6)
Solution quality of executing the algorithm on different multi-processor systems

Test case size	Number of processors	GA				
		C2	C5	C10	C14	C7
		fitness $\pm$ STD	fitness $\pm$ STD	fitness $\pm$ STD	fitness $\pm$ STD	fitness $\pm$ STD
10	5	40 $\pm$ 0.00	40 $\pm$ 0.00	40 $\pm$ 0.0	40 $\pm$ 0.00	40 $\pm$ 0.00
10	7	40 $\pm$ 0.00	40 $\pm$ 0.00	40 $\pm$ 0.0	40 $\pm$ 0.00	40 $\pm$ 0.00
10	9	40 $\pm$ 0.00	40 $\pm$ 0.00	40 $\pm$ 0.0	40 $\pm$ 0.00	40 $\pm$ 0.00
15	5	63 $\pm$ 0.00	63 $\pm$ 0.00	63 $\pm$ 0.0	63 $\pm$ 0.00	63 $\pm$ 0.00
15	7	63 $\pm$ 1.09	63 $\pm$ 0.00	63 $\pm$ 0.0	63 $\pm$ 0.00	63 $\pm$ 0.00
15	9	63 $\pm$ 0.00	63 $\pm$ 0.00	63 $\pm$ 0.0	63 $\pm$ 0.00	63 $\pm$ 0.00
30	5	82 $\pm$ 0.00	82 $\pm$ 0.00	82 $\pm$ 0.0	82 $\pm$ 0.00	82 $\pm$ 0.00
30	7	82 $\pm$ 0.00	82 $\pm$ 0.00	82 $\pm$ 0.0	82 $\pm$ 0.00	82 $\pm$ 0.00
30	9	82 $\pm$ 0.00	82 $\pm$ 0.00	82 $\pm$ 0.0	82 $\pm$ 0.00	82 $\pm$ 1.09
50	5	101 $\pm$ 1.09	101 $\pm$ 1.09	101 $\pm$ 0.0	101 $\pm$ 1.09	103 $\pm$ 4.60
50	7	101 $\pm$ 0.00	102 $\pm$ 3.07	101 $\pm$ 0.0	102 $\pm$ 2.42	111 $\pm$ 13.33
50	9	101 $\pm$ 1.52	102 $\pm$ 2.77	101 $\pm$ 0.0	101 $\pm$ 1.09	106 $\pm$ 7.59
75	5	59 $\pm$ 0.00	61 $\pm$ 4.80	59 $\pm$ 0.00	62 $\pm$ 4.89	70 $\pm$ 8.48
75	7	59 $\pm$ 0.00	61 $\pm$ 3.39	59 $\pm$ 0.00	60 $\pm$ 2.80	69 $\pm$ 8.92
75	9	59 $\pm$ 0.00	63 $\pm$ 6.43	59 $\pm$ 0.00	61 $\pm$ 5.70	69 $\pm$ 8.22
100	5	139 $\pm$ 3.23	153 $\pm$ 26.04	138 $\pm$ 0.00	144 $\pm$ 10.68	173 $\pm$ 32.72
100	7	139 $\pm$ 2.92	152 $\pm$ 23.02	138 $\pm$ 1.52	153 $\pm$ 19.18	180 $\pm$ 33.50
100	9	139 $\pm$ 2.61	150 $\pm$ 20.03	138 $\pm$ 0.00	140 $\pm$ 17.9	163 $\pm$ 33.8

5.4 Results and Discussion of Cuckoo Search

The conducted experiments along with the discussion of the empirical results are presented in this section. A thorough study of the algorithm parameters is provided.

5.4.1 Parameter Settings

Experimentation was done with the parameter settings given in Table (5.7). Based on these parameters, a number of combinations were formed as given in Table (5.8) and were used for parameter sensitivity analysis.

Table (5.7)
Parameter settings used in CS

Number of nests	20, 40, 60
λ	random between (0-1), 0.25, 0.5, 0.75, 1.0
p_a	random between (0-0.5), random between (0- 0.75)

Table (5.8)
Parameter combinations used in CS

combination	Parameters	
	λ	p_a
Cc1	0.25	random between (0-0.5)
Cc2	0.5	random between (0-0.5)
Cc3	0.75	random between (0-0.5)
Cc4	1.0	random between (0-0.5)
Cc5	0.25	random between (0- 0.75)
Cc6	0.5	random between (0- 0.75)
Cc7	0.75	random between (0- 0.75)
Cc8	1.0	random between (0- 0.75)
Cc9	random between (0-1)	random between (0-0.5)
Cc10	random between (0-1)	random between (0- 0.75)

5.4.2 Effect of Parameter Values

Testing the performance of the modified version of cuckoo search was applied using a similar approach used in testing GA. The process of testing in our work starts by performing parameter sensitivity analysis. Test case 1 and Test case 2 are used to study the effect of changing the

parameters such as number of nests and values of λ and p_a on the algorithm performance. This was followed by further experimentation on test cases using the identified best parameters combination as will be discussed in later sections.

The effect of different CS parameters on the quality of solution and run time is discussed below. These include the effects of number of nests, λ and p_a . The test cases considered for sensitivity analysis are Test case1 (# of tasks 10) and Test case2 (# of tasks 15) on a multi-processor system composed of 5 processors.

5.4.2.1 Effect of Number of Nests

Results shown in Figure (5.10) and tables (5.9) and (5.10) indicate that the algorithm produced the best results in terms of solution quality when number of nests=60 for almost all combinations we tried.

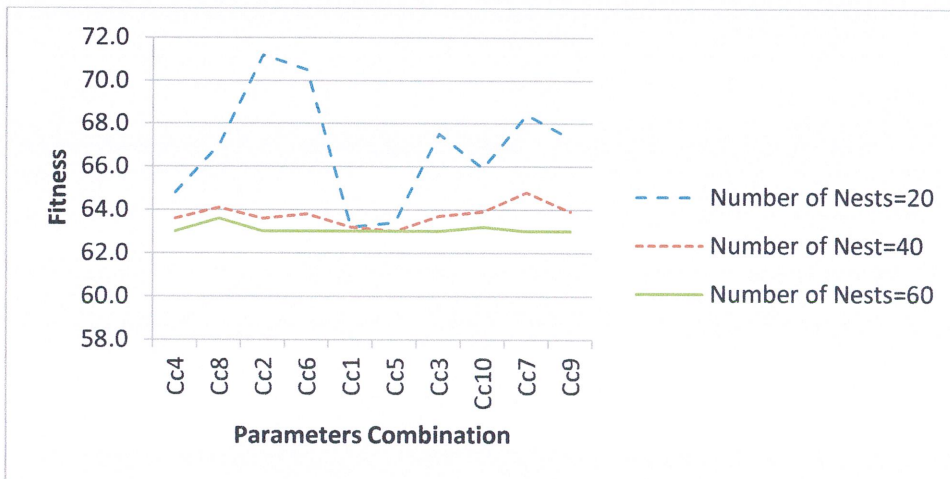


Figure (5.10)
CS algorithm performance in terms of solution quality for Test case 2

Table (5. 9)
Cuckoo search results for Test case1 (# of tasks 10)

	Cc8		Cc6		Cc10		Cc5		Cc7	
Number of nests	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)
Number of nests=20	40	38	41	35	40	36	40	88	40	37
Number of nests=40	40	64	40	60	40	61	40	141	40	62
Number of nests=60	40	89	40	84	40	85	40	211	40	87
	Cc3		Cc9		Cc2		Cc1		Cc4	
Number of nests	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)
Number of nests=20	41	44	40	43	41	40	40	40	40	42
Number of nests=40	40	78	40	73	40	68	40	68	40	70
Number of nests=60	40	112	40	97	40	97	40	97	40	99

Table (5. 10)
Cuckoo search results for Test case 2 (# of tasks 15)

	Cc4		Cc2		Cc1		Cc3		Cc9	
Number of nests	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)
Number of nests=20	64.8	56	71.2	56	63.2	60	67.5	65	67.3	65
Number of nests=40	63.6	98	63.6	98	63.2	110	63.7	115	63.9	109
Number of nests=60	63.0	138	63.0	137	63.0	154	63.0	154	63.0	154
	Cc8		Cc6		Cc5		Cc10		Cc7	
Number of nests	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)	Avg fitness	Avg. Time (min)
Number of nests=20	67.0	50	70.5	60	63.4	54	65.9	59	68.4	67
Number of nests=40	64.1	86	63.8	104	63.0	100	63.9	106	64.8	116
Number of nests=60	63.6	122	63.0	149	63.0	137	63.2	151	63.0	159

Figure (5.11) illustrate the effect of the number of nests on fitness function while using Test case 2 for $\lambda = 0.5$, $P_a = \text{random value (0-0.5)}$ on 5 processors system. The results show that the fitness produced by 30 independent runs varied significantly with 20 nests, while 60 nests produced best quality solutions with least variation as can be seen in Figure (5.11).

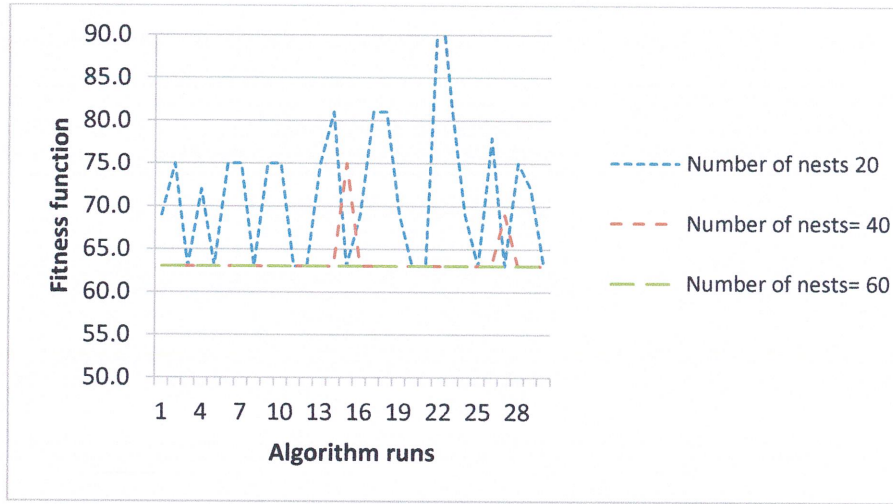


Figure (5.11)
Fitness function values obtained during 30 runs
for 20, 40, and 60 nests (Results for Test case 2 on the combination Cc2 on 5
processors system)

5.4.2.2 Effect of λ and p_a Values

The effect of λ and p_a was investigated while keeping the number of nests = 60, since this number generated the best results as mentioned in the previous section. Referring to Figure (5.10) and Tables (5.9) and (5.10), it is observed that the best quality solutions were obtained with combinations Cc4, Cc2, Cc1, Cc3, Cc9, Cc6, Cc5 and Cc7.

The above combinations were subjected to testing for run time using test case 2. Figure (5.12) depicts the results obtained which indicated that

combinations Cc4, Cc2, and Cc5 were most efficient with regard to the run time (and quality of solution as well).

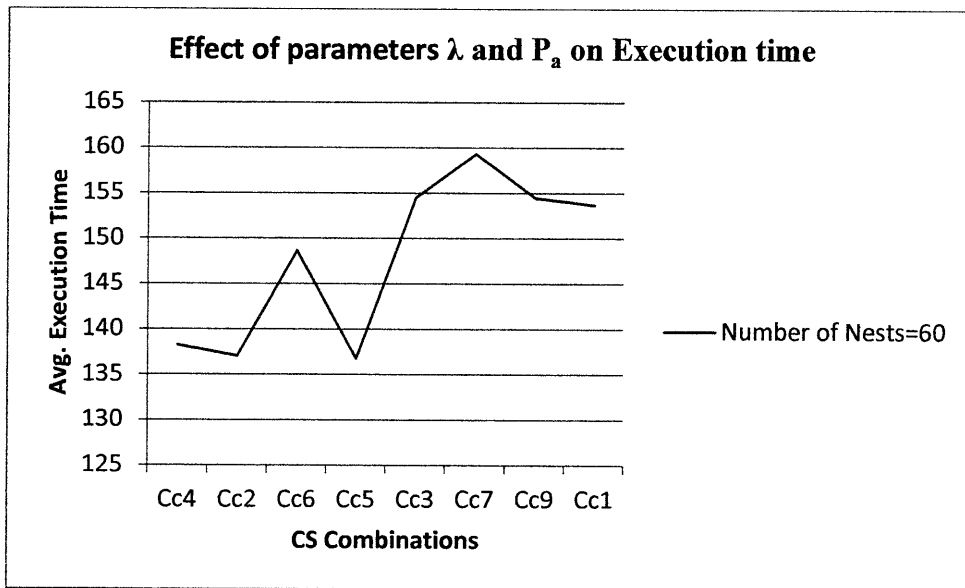


Figure (5.12)
CS performance in terms of average execution time

5.4.3 Experimentation with Selected Parameter Combinations

Based on above results, 60 nests and combinations Cc4, Cc2, and Cc5 were used for further experimentation with other test cases. The results are shown in Table (5.11) with regard to the quality of solution for all test cases. The results show for small test cases (10, 15, and 30 tasks), the quality of solution was almost the same for all combinations used. However, when the test cases size increased to 50, 75, and 100 tasks, the effect of parameter values on the quality of solution was more significant. Results in Table (5.11) suggest that combination Cc5 was most efficient in terms of solution quality in general.

Table (5.11)
Solution quality analysis of best parameter combinations

Number of tasks	Number of processors	Cuckoo		
		Cc4	Cc2	Cc5
		fitness±STD	fitness±STD	fitness±STD
10	5	40±0.00	40±0.00	40±0.00
10	7	40±0.00	40±0.00	40±0.00
10	9	40±0.00	40±0.00	40±0.00
15	5	63±0.00	63±0.00	63±0.00
15	7	63±1.52	63±0.00	63±0.00
15	9	63±0.00	63±0.00	63±0.00
30	5	86±6.08	83±3.76	82±0.00
30	7	83±2.75	83±3.31	82±1.73
30	9	85±5.92	83±3.11	82±0.00
50	5	111±9.55	105±5.79	102±2.38
50	7	108±6.36	106±6.44	102±2.28
50	9	108±8.21	101±0.00	103±4.52
75	5	66±5.76	62±3.09	62±2.99
75	7	66±5.09	63±3.98	62±2.95
75	9	83±13.30	64±4.19	60±1.89
100	5	165±11.10	156.1±12.93	144±8.11
100	7	163±12.10	155.7±11.61	148±10.30
100	9	153±11.10	147.9±7.60	138±0.00

With regard to the time analysis, results in Table (5.12) indicate that in general, combination Cc4 was most efficient. The observations was made based on the time it took the algorithm to reach the best solution.

Table (5. 12)
Time to reach best solution quality for selected combinations (in minutes)

Number of tasks	Number processors	Cuckoo		
		Cc4	Cc2	Cc5
		Time (min)	Time (min)	Time (min)
10	5	1	1	3
10	7	1	1	2
10	9	1	1	2
15	5	3	6	5
15	7	4	4	5
15	9	3	3	6
30	5	21	12	15
30	7	11	9	10
30	9	9	9	11
50	5	13	18	29
50	7	16	30	22
50	9	20	30	24
75	5	27	19	47
75	7	37	26	50
75	9	7	28	48
100	5	33	38	47
100	7	44	25	60
100	9	16	30	68

5.5 Results and Discussion of Cuckoo Search with Penalty

The cuckoo search algorithm discussed earlier discards the infeasible solution. In this variant of CS, infeasible solutions are kept but with added penalty. The procedure includes accepting up to 5% of the total number of nests. The proposed cuckoo search with penalty was evaluated with the different test cases as discussed below. We studied the effect of changing the parameters such as number of nests and values of λ and p_a on Test case 1 and Test case 2. The results governed further

experimentation conducted on other test cases, as will be discussed in later sections.

5.5.1 Parameter Settings

Settings used in Section 5.4.1 for CS algorithm were also used for experimentation here.

5.5.2 Effect of parameter Values

The effect of different parameters on the quality of solution and run time is discussed below. These include the effects of number of nests, λ and p_a . Test case1 and test case2 were considered for sensitivity analysis on a multi-processor system composed of 5 processors.

5.5.2.1 Effect of Number of Nests

Results shown in Figure (5.13) and Tables (5.13) and (5.14) show that the algorithm produced the best results in terms of solution quality while using test case 1 and test case 2 with 60 nests for almost all combinations.

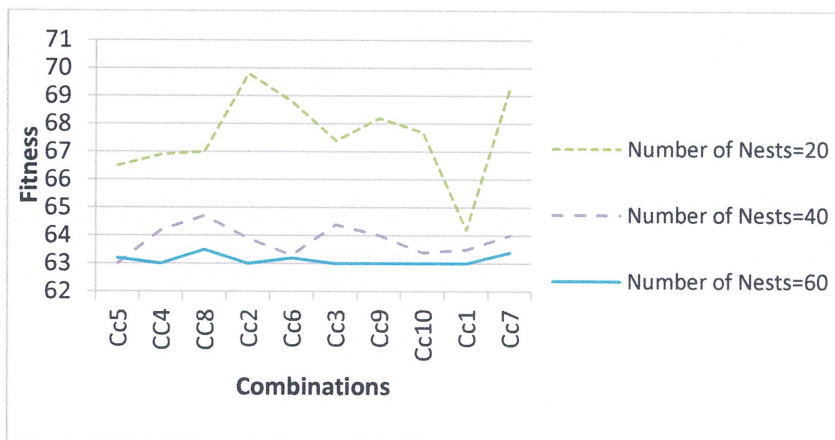


Figure (5.13)

Effect of number of nests on algorithm performance in terms of solution quality on Test case 2

Table (5. 13)
Cuckoo search with penalty results Test case 1 (# of tasks 10)

Number of nests	Cc8		Cc5		Cc10		Cc6		Cc7	
	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)
20	40.0	44	40.0	43	40.0	46	40.0	40	40.2	36
40	40.0	78	40.0	72	40.0	82	40.0	67	40.0	61
60	40.0	112	40.0	102	40.0	117	40.0	95	40.0	85
Number of nests	Cc4		Cc1		Cc3		Cc9		Cc2	
	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)
20	40.2	41	40.0	41	40.0	44	40.0	40	40.2	39
40	40.0	68	40.0	70	40.0	79	40.0	68	40.0	66
60	40.0	95	40.0	100	40.0	113	40.0	95	40.0	94

Table (5. 14)
Cuckoo search with penalty results Test case 2 (# of tasks 15)

Number of nests	Cc5		Cc8		Cc6		Cc10		Cc7	
	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)
20	66.5	56	67	58	68.8	58	67.7	53	69.3	62
40	63	99	64.7	101	63.3	100	63.4	89	64	107
60	63	134	63.5	141	63.2	141	63	125	63.4	152
Number of nests	Cc1		Cc4		Cc2		Cc9		Cc3	
	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)	Avg. fitness	Avg. Time (min)
20	64.2	59	66.9	57	69.8	65	68.2	62	67.4	53
40	63.5	107	64.2	100	63.9	115	64	106	64.4	92
60	63	152	63	142	63	150	63	164	63.5	130

Results in Figure (5.14) demonstrate the effect of the number of nests on the objective function while using Test case 2 for $\lambda = 0.5$, $P_a =$ random value (0-0.5) on 5 processors system. The results show

significant variations in the fitness produced by 30 independent runs with 20 nests, while variation in the solution quality was least with 60 nests.

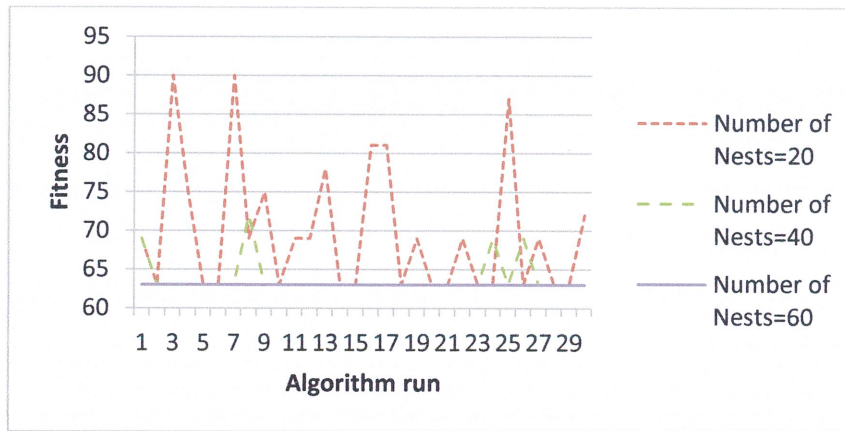


Figure (5.14)

Effect of number of nests on CS with penalty algorithm performance in terms of Solution quality for Test case 2 with combinationCc2 on 5processors

5.5.2.2 Effect of λ and p_a Values

The number of nests is kept fixed of size= 60 in this stage to investigate the effect of λ and p_a , since this number generated the best results as mentioned in the previous section. The best quality solutions were obtained with combinations Cc4, Cc2, Cc3, Cc9, Cc1 and Cc10 by referring to Figure (5.15) and Tables (5.13) and (5.14).

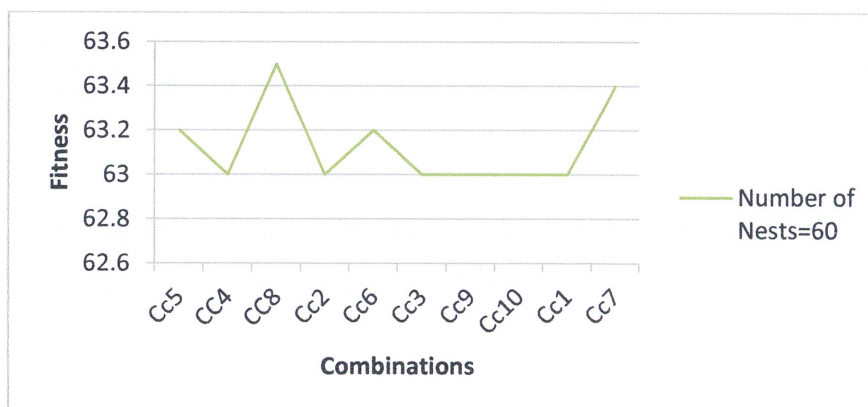


Figure (5.15)

Effect of parameter combinations on the performance in terms of Solution quality on Test case 2

The above combinations were subjected to testing for run time using test case 2 and Test case 1. Figure (5.16) depicts the results obtained which indicated that combinations Cc4, Cc2 and Cc10 were most efficient with regard to the run time (and quality of solution as well).

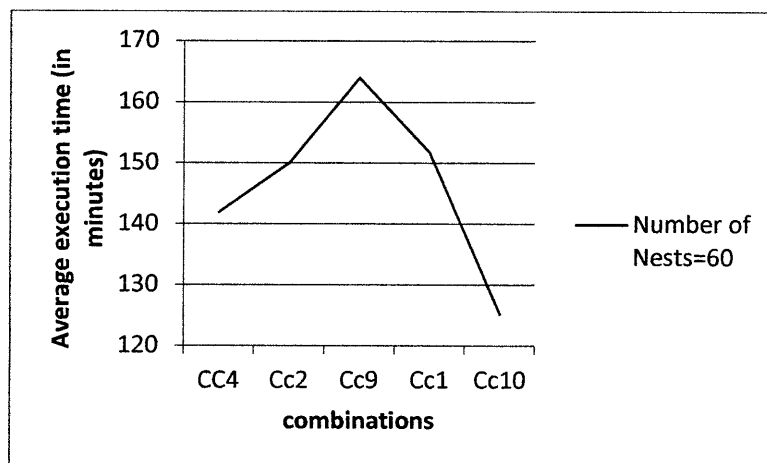


Figure (5.16)

Effect of parameter combinations on the performance in terms of Average execution time on Test case 2

5.5.3 Experimentation with Selected Parameter Combinations

Combinations Cc4, Cc2, and Cc10 and 60 nests were used for further experimentation on other test cases based on the previous observations. Table (5.15) illustrates the results with regard to the quality of solution for all test cases. The results show that the quality of solutions was almost the same for all combinations used on test cases 1, 2 and 3 (# of tasks 10, 15 and 30 respectively). Furthermore, when the size increased to 50, 75, and 100 tasks, the results indicate that the effect of parameter values on the quality of solution was more significant. In general, combination Cc10 is observed to be the best in terms of solution quality, as shown in the Table (5.15).

Table (5. 15)
Solution quality analysis for Test case1 with best selected parameter combinations

Number of tasks	Number of processors	CS with penalty		
		Cc4	Cc2	Cc10
		fitness±STD	fitness±STD	fitness±STD
10	5 processor	40±0.0	40±0.0	40±0.0
10	7 processor	40±0.0	40±0.0	40±0.0
10	9 processor	40±0.0	40±0.0	40±0.0
15	5processor	63±0.0	63±0.0	63±0.0
15	7processor	63±0.0	63±0.0	63±2.19
15	9processor	64±2.74	63±1.64	63±0.0
30	5processor	84±3.4	84±4.27	82±0.0
30	7processor	85±4.68	82±0.0	82.9±2.7
30	9processor	84±3.32	84.4±2.38	82±2.69
50	5 processor	102±2.73	104±4.63	101±0.0
50	7 processor	106±6.02	104±4.92	107±7.68
50	9 processor	108±7.45	104±5.14	104.5±4.73
75	5 processor	65±4.75	62±3.94	62±2.14
75	7 processor	67±5.35	64±4.5	63±3.44
75	9 processor	67±5.16	63±3.35	62±4.07
100	5 processor	161±16	152±12.3	157.1±12.72
100	7 processor	160±12	151±10.8	152.9±7.89
100	9 processor	160±12.7	148±11.3	151.2±13.04

With respect to the time analysis, results in Table (5.16) illustrate the time it took the algorithm to reach the best solution. It shows that in general, combination Cc10 was efficient in most tested cases. However, combination Cc10 degrades for larger test cases (number of tasks 75 and 100) when it is compared to combination Cc4.

Table (5. 16)
Time to achieve best results (in minutes)

Number of tasks	Number of processors	CS with penalty		
		Cc4	Cc2	Cc10
		Time (min)	Time (min)	Time (min)
10	5	1	1	1
10	7	1	2	1
10	9	1	1	1
15	5	8	5	4
15	7	5	4	5
15	9	9	5	8
30	5	9	12	10
30	7	15	9	11
30	9	11	9	8
50	5	28	26	18
50	7	26	25	15
50	9	21	20	18
75	5	21	45	25
75	7	21	30	25
75	9	35	34	29
100	5	23	38	28
100	7	28	37	39
100	9	25	40	39

5.6 Comparison of Algorithms

Results obtained from best combinations of the three algorithms were mutually compared with regard to quality of solutions and time efficiency, for which the details follow.

5.6.1 Solution Quality Analysis

Table (5.17) presents a comparison of the performance of the three algorithms obtained using the best parameter combination for each algorithm discussed in the previous sections. Results show that for

smaller test case sizes, both cuckoo search and cuckoo search with penalty achieved the same level of performance as that of genetic algorithm with respect to the quality of solutions. However, for larger test cases (Number of tasks 75 and 100) CS and CS with penalty was not able to produce the best solution obtained using GA (for the combination C10).

Table (5. 17)
Solution quality comparison

Number of tasks	Number of processors	GA_C10	Cuckoo_Cc5	CS with Penalty_Cc10	Difference between GA and Cuckoo	Difference between GA and Cuckoo with penalty
10	5	40 ± 0.0	40±0.0	40±0.0	0.0	0.0
10	7	40 ± 0.0	40±0.0	40±0.0	0.0	0.0
10	9	40 ± 0.0	40±0.0	40±0.0	0.0	0.0
15	5	63± 0.0	63±0.0	63±0.0	0.0	0.0
15	7	63± 0.0	63±0.0	63±2.19	0.0	0.0
15	9	63± 0.0	63±0.0	63±0.0	0.0	0.0
30	5	82 ± 0.0	82±0.0	82±0.0	0.0	0.0
30	7	82 ± 0.0	82±1.73	82.9±2.7	0.0	0.9
30	9	82 ± 0.0	82±0.0	82±2.69	0.0	0.0
50	5	101± 0.0	102±2.38	101±0.0	1.0	0.0
50	7	101± 0.0	102±2.28	107±7.68	1.0	6.0
50	9	101± 0.0	103±4.52	104.5±4.73	2.0	3.0
75	5	59± 0.0	62±2.99	62±2.14	3.0	3.0
75	7	59± 0.0	62±2.95	63±3.44	3.0	4.0
75	9	59± 0.0	60±1.89	62±4.07	1.0	3.0
100	5	138± 0.0	144±8.11	157±12.72	6.0	19.0
100	7	138±1.52	148±10.3	152±7.89	10.0	14.0
100	9	138±0.0	138± 0.0	151±13.04	0.0	13.0

5.6.2 Comparison of Run Time

The performance of GA, CS and CS with penalty was compared in terms of time efficiency using two approaches. We used average time as displayed in Table (5.18) and using the time required by each algorithm to reach the best solutions as presented in Table (5.19). The latter approach provides a better comparison base for the algorithms considered since different settings of population sizes are used for each algorithm. Results in both tables show how GA outperforms CS and CS with penalty in terms of execution time. However, CS variations using specific combinations of parameters show a comparable performance to GA for smaller test cases (number of tasks 10, 15 and 30 tasks) as depicted in Table (5.19) and Figure (5.17). In general, observing results illustrated in Table (5.19) show that GA performs better than CS and CS with penalty in terms of execution time to find best solution. GA produced results in shorter time compared to CS in a percentage ranging between (31%-58%) and GA outperforms CS with penalty in a percentage ranging (25% - 50%).

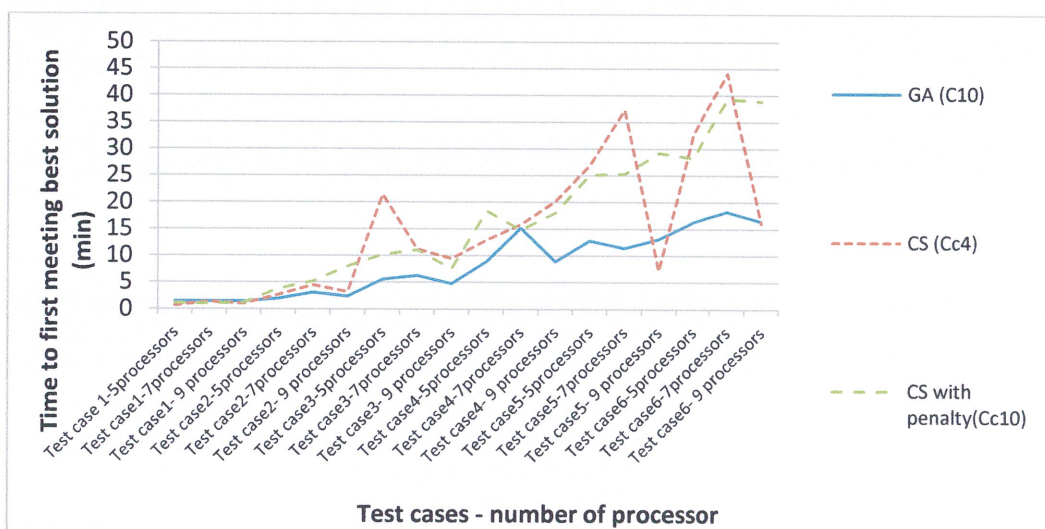


Figure (5. 17)
Best combinations of GA, CS and CS with penalty comparison time analysis
(in minutes)

Table (5.18)
Performance analysis of average execution time (in minutes)

Number of tasks	Number of processors	GA					Cuckoo			penalty		
		C2	C5	C10	C14	C7	Cc4	Cc2	Cc5	Cc4	Cc2	Cc10
		AVG Time	AVG Time	AVG Time	AVG Time	AVG Time	AVG Time	AVG Time	AVG Time	AVG Time	AVG Time	AVG Time
10	5	6	10	7	11	10	99	97	211	95	94	117
10	7	10	18	11	21	17	94	233	94	96	222	89
10	9	9	16	10	18	16	90	103	95	88	95	98
15	5	7	15	8	14	15	138	137	137	142	164	125
15	7	15	22	16	28	23	165	124	124	157	165	124
15	9	14	23	14	25	27	155	131	141	131	156	172
30	5	16	31	18	30	31	166	193	195	180	217	191
30	7	16	28	22	33	32	176	179	167	188	193	190
30	9	16	27	19	32	27	176	165	181	185	159	191
50	5	21	32	22	37	34	204	201	241	227	264	264
50	7	28	66	28	52	61	259	207	210	260	198	249
50	9	18	34	22	36	32	203	225	224	255	255	254
75	5	22	39	28	45	39	282	281	280	308	563	264
75	7	26	40	26	44	40	236	267	260	262	288	316
75	9	27	38	27	40	38	379	286	281	263	237	280
100	5	26	43	34	46	43	308	310	302	401	307	315
100	7	31	48	34	52	48	359	297	277	319	300	343
100	9	26	42	33	47	40	345	372	492	287	328	326

Table (5. 19)
Time required by each algorithm to produce the best solution quality (in minutes)

Number of tasks	Number of processors	GA					CS			CS with penalty		
		C2	C5	C10	C14	C7	Cc4	Cc2	Cc5	Cc4	Cc5	Cc10
		Time	Time	Time	Time	Time	Time	Time	Time	Time	Time	Time
10	5	1	2	1	2	3	1	1	3	1	1	1
10	7	2	3	1	4	4	1	1	2	1	2	1
10	9	2	3	1	3	4	1	1	2	1	1	1
15	5	2	7	2	5	8	3	6	5	8	5	4
15	7	3	6	3	7	8	4	4	5	5	4	5
15	9	4	6	2	7	9	3	3	6	9	5	8
30	5	5	15	6	13	18	21	12	15	9	12	10
30	7	5	14	6	14	19	11	9	10	15	9	11
30	9	5	13	5	12	15	9	9	11	11	9	8
50	5	9	20	9	24	28	13	18	29	28	26	18
50	7	18	58	15	38	57	16	30	22	26	25	15
50	9	9	25	9	21	28	20	30	24	21	20	18
75	5	12	32	13	35	35	27	19	47	21	45	25
75	7	13	33	11	32	35	37	26	50	21	30	25
75	9	13	33	13	31	35	7	28	48	35	34	29
100	5	14	36	16	36	41	33	38	47	23	38	28
100	7	18	43	18	45	44	44	25	60	28	37	39
100	9	15	42	16	47	40	16	30	68	25	40	39

5.6.3 Percentage Saving in Power

The amount of power saving can be seen in Table (5.20) and Figures (5.18) and (5.19) where all test cases were executed on the highest voltage level provided by the processors. The best results obtained along with using any of the algorithms are used to measure the amount of saving. To measure the amount of energy saving then the results obtained of executing tasks using DVS are compared with those produced when processing the tasks using the highest voltage a processor can operate on.

Saving in best cases reached approximately 89% using GA as adapted and using DVS to control energy the processor system. With regard to the two variants of cuckoo search, a similar performance in terms of power saving was observed. This can be attributed to the nature of data graphs used in the experimentation stage and the way in which the algorithms were adapted.

Table (5.20)
Comparison of percentage saving using 5 processors

Number of tasks	Energy consumption of execution tasks on maximum voltage level	% Energy Saving		
		GA	CS	CS with penalty
10	360	88.89	88.89	88.89
15	342	81.58	81.58	81.58
30	400	79.5	79.5	79.5
50	432	76.62	76.39	76.62
75	490	87.96	87.35	87.35
100	566	75.62	74.56	73.14

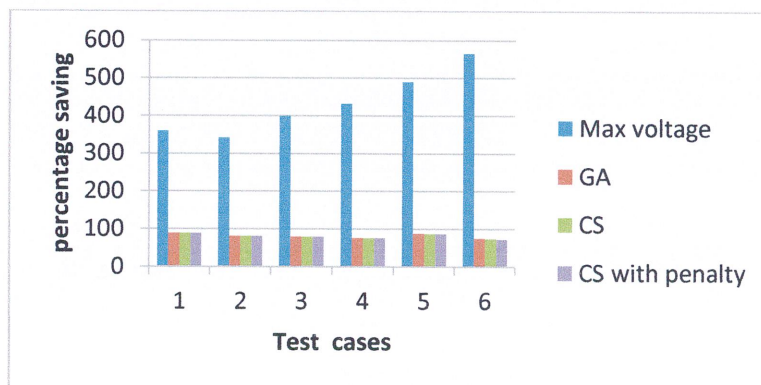


Figure (5.18)
Percentage of Energy saving

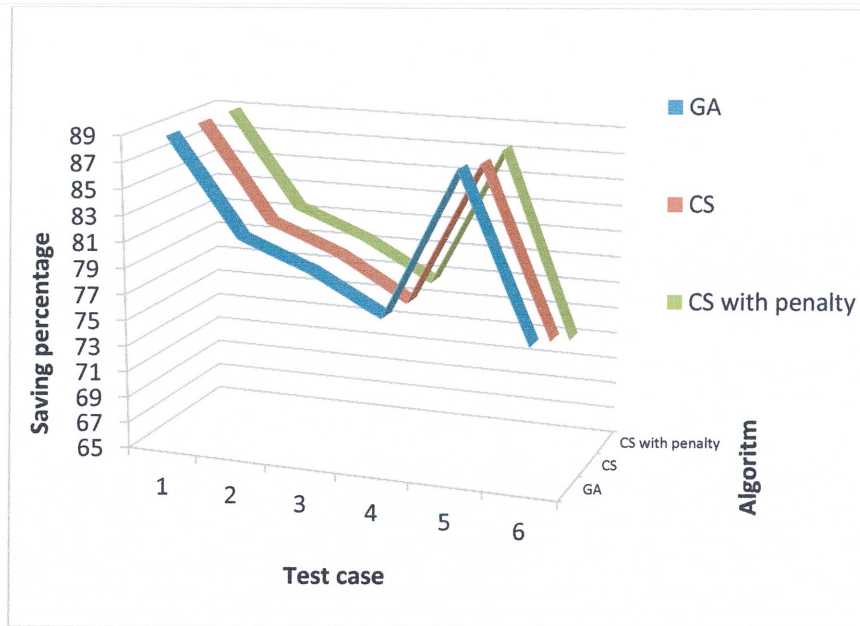


Figure (5. 19)

Percentage saving comparison of Energy consumption (using 5 processors)

5.6.4 Statistical Test Results

The results obtained had been tested with Wilcoxon ranked-sum test. The Wilcoxon rank sum test is an example of a non-parametric or distribution free test to test the null hypothesis that the median of a distribution is equal to some value. The test is used for analyzing two sets of ordered categorical data where a numerical scale is inappropriate but it is possible to rank the observations (Dagaard, 2008).

The following table shows the results of comparing genetic algorithm and two variants of cuckoo search. The results for GA reported here are for best parameter combination (C10), while the best parameter combinations of cuckoo search and cuckoo search with penalty were Cc4 and Cc10 respectively. The test depends on ranking the absolute difference between the data sets to be compared. The respective p-values are reported in Table (5.21).

Table (5. 21)
P-level value of wilcoxon statistical test

Test Case	of Number processors	CS			CS with Penalty		
		Cc5	Cc4	Cc2	Cc2	Cc4	Cc10
Test case 1	5	0.00	0.00	0.00	0.00	0.00	0.00
	7	0.00	0.00	0.00	0.00	0.00	0.00
	9	0.00	0.00	0.00	0.00	0.00	0.00
Test case 2	5	0.00	0.00	0.00	0.00	0.00	0.00
	7	0.00	0.00	0.00	0.00	0.00	0.00
	9	0.00	0.00	0.00	0.00	0.00	0.00
Test case 3	5	0.00	0.00	0.07	0.02	0.02	0.00
	7	0.00	0.01	0.07	0.00	0.02	0.05
	9	0.00	0.01	0.07	0.07	0.02	0.00
Test case 4	5	0.07	0.00	0.00	0.01	0.03	0.00
	7	0.00	0.00	0.00	0.01	0.00	0.00
	9	0.02	0.00	0.00	0.01	0.00	0.00
Test case 5	5	0.00	0.00	0.00	0.00	0.00	0.00
	7	0.00	0.00	0.00	0.00	0.00	0.00
	9	0.00	0.00	0.00	0.00	0.00	0.00
Test case 6	5	0.00	0.00	0.00	0.00	0.00	0.00
	7	0.00	0.00	0.00	0.00	0.00	0.00
	9	0.04	0.00	0.00	0.00	0.00	0.00

5.7 Effect of Changing the Number of Processors on Algorithm Performance

The increase of number of processors did not affect the solution quality significantly as observed in Table (5.17). Figures (5.20) to (5.22) show results obtained from running the algorithms on multiprocessor systems composed of 5, 7 and 9 processors. The results are not showing a significant change in the running timing scored except for some test cases which could be attributed to the nature of test data used.

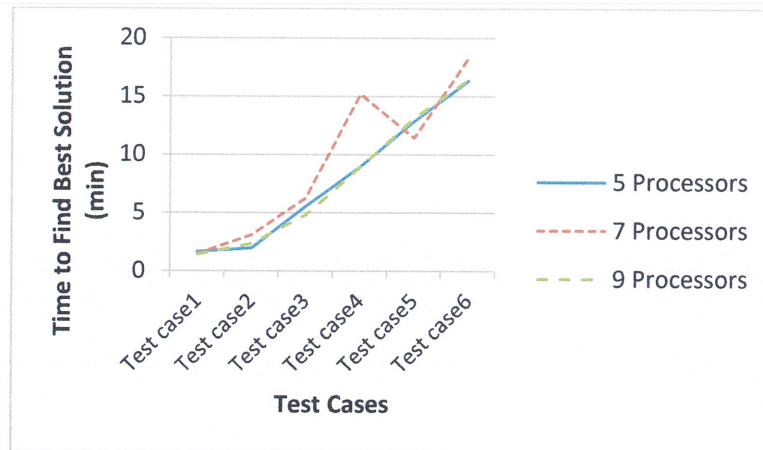


Figure (5. 20)

GA combination C10 performance on different multiprocessor systems (composed of 5, 7, and 9 processors)

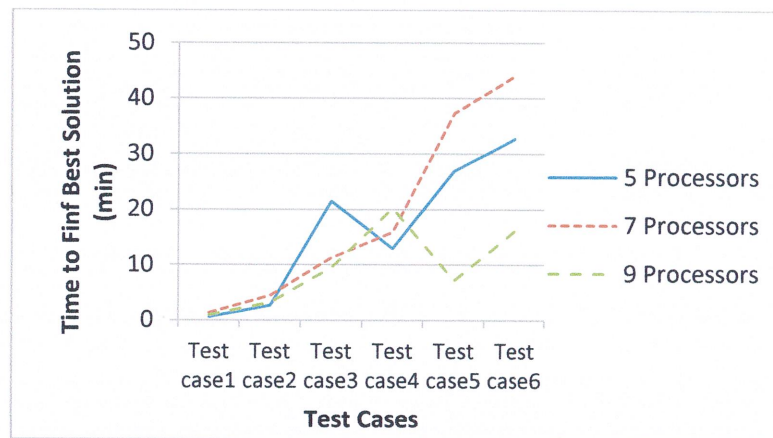


Figure (5. 21)

CS combination Cc4 performance on different multiprocessor systems (composed of 5, 7, and 9 processors)

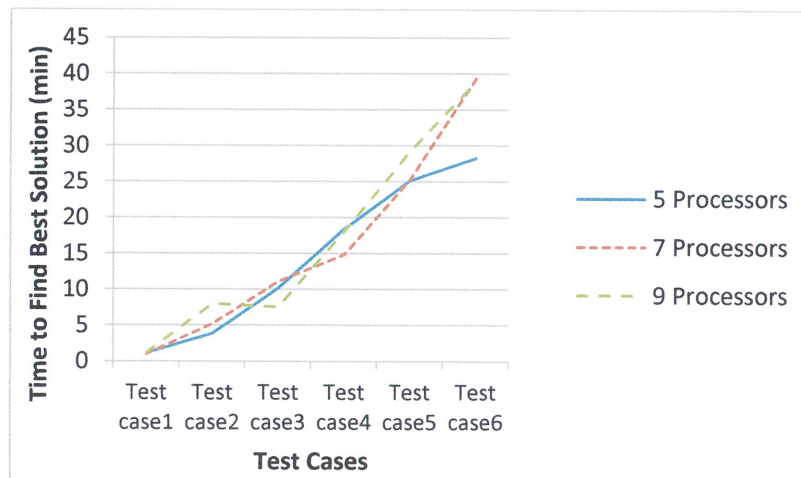


Figure (5. 22)

CS with penalty combination Cc10 performance on different multiprocessor systems (composed of 5, 7, and 9 processors)

5.8 Summary

Sensitivity analysis leads to an appropriate selection of parameters for both GA and cuckoo search algorithms. The detailed experimentation applied to data graphs provides an insight into the performance of the studied algorithms with regard to solution quality and execution time.

In case of GA, the effect of a number of parameter was tested. These parameters include population size, and the different types of crossover and mutation. In contrast, cuckoo search has relatively less parameters to test and tune. These include the step size which is involved in Lévy flight operation. The Lévy flight operation was controlled by two variables, namely, λ and p_a in order to allow the search not to rely entirely on random walks, and to gain fast convergence.

In terms of quality of solutions, the adapted GA, CS, and CS with penalty produced the similar quality of results. Furthermore, given ample computation time, CS was always able to find the optimal solution. More specifically, the two variants of cuckoo search displayed better results in terms of solution quality and run time with smaller test cases, while the performance degraded with larger task graphs.

Chapter Six

Conclusion and Future Work

6.1 Conclusion	111
6.2 Future work	113

Chapter Six

Conclusion and Future Work

6.1 Conclusion

Energy consumption is a design issue for battery-operated devices such as personal digital assistants, digital camcorders and mobile phones. Designing an efficient system with lower power consumption is important where power supply is a critical issue. Reducing power consumption or managing energy saving in a system can be achieved in various ways. Many hardware techniques such as dynamic power management (DPM) and dynamic voltage scaling (DVS) have been proposed.

This study discourses the design of problem of energy aware scheduling of real-time tasks on DVS multi-processor systems. I model the real-time task allocation as an optimization problem subject to a number of constraints. Two population-based iterative algorithms (GA and CS) have been adapted to obtain good quality solutions. I proposed a two-dimensional solution encoding scheme for efficient application of the GA and CS algorithms. A number of variations of crossover and mutation operators for GA have been proposed. For CS, methods for determining suitable value for its parameters (i.e. λ , p_a) were proposed. I also proposed a variation of basic CS (called CS with penalty) that allows infeasible solutions to be part of the population with a penalty. Such solutions have the potential to become feasible solution in the subsequent iteration and may also lead to better quality solution.

The adapted algorithms have been extensively studied through simulation. A comprehensive parameter sensitivity analysis was carried out to determine most suitable parameters' value and combinations. The simulation results for the proposed/adapted algorithms for a number of test-cases have been presented and compared. Simulation results show that GA and CS give the same quality results for most of the test cases. However, GA outperforms the CS in terms of run-time. That is, GA was able to find (sub) optimal solutions in less than as compared to CS. The simulation results also show that by scheduling the task on the suitable processors at a suitable voltage level can achieve a saving of up to 89% in energy consumption.

Simulation results of the GA variations including engineering crossover, mutation and replacement operators provided clearer view of which types of crossover/ mutation and relative rates might be deployed to produce better results.

Cuckoo search as has been adapted to solve the allocation problem on multiprocessor system for discrete domains is efficient and easy to have the effect of its parameters analyzed and then tuned. Simulation results of the proposed variation of Cuckoo search with penalty showed a very promising which is simple and efficient.

It is worth to mention that one of the limitations of the algorithm considered herein is that it is offline algorithm for static scheduling and would not be work efficiently for online and dynamic scheduling.

Time limited our chance to include more test cases; more test cases of larger size could be considered to enhance the research outcome and considering larger sizes of the DAGs would give a better indicator of the

effect of changing the number of processors on algorithms performance. Two different rates of crossover and mutation operators were considered in testing, however testing the effect of more possibilities of the corresponding rates of both operators might refine our results obtained.

The simulation has been done for homogenous multi-processor systems where each processor operates at a fixed number of voltage levels. The research also can be extended to study the effect of using heterogeneous system on the overall performance of the adapted algorithms.

6.2 Future work

The work presented in thesis can be extended in a number of directions. One area that needs further exploration is to design adaptive GA that dynamically selects the most suitable crossover and mutation types in each iteration. As we used fixed values for the parameters of CS, effect of dynamically changing the value of the parameters can be studied. As a number of previous studies show that the iterative algorithms could provide better solution when they hybridized with other algorithm. As a future work, developing hybrid versions of GA and CS can be explored. Since I conducted all the experimentations for homogeneous processor, another possible future work could be to study the performance of the proposed algorithms for heterogeneous DVS multi-processor systems.

One of the assumptions considered for the adapted versions of cuckoo search in this work involve having one egg (solution) per nest (Yang & Deb, 2009). Changing this assumption to permit having more

than one egg (solution) per nest will enhance the performance by widening the search space and increasing the possibilities of solution modification and producing multiple solutions. This may influence the speed of algorithm convergence to generate solutions. Another enhancement is to be considered is related to the adaptation approach of cuckoo search with penalty algorithm. Adding penalty to CS algorithm tend to be a promising approach in solving optimization problems. More experimentation and research in this area is required to study the effect of changing the penalty percentage on the algorithm performance. Experimentation may be extended to determine the best value to be used to enhance the performance of the algorithm.

References

- Alahmad, B. N., & Gopalakrishnan, S. (2011). Energy efficient task partitioning and real-time scheduling on heterogeneous multiprocessor platforms with qos requirements. *Sustainable Computing: Informatics and Systems*, 1(4), 314-328.
- Alajmi, A., & Wright, J. (2014). Selecting the most efficient genetic algorithm sets in solving unconstrained building optimization problem. *International Journal of Sustainable Built Environment*, 3(1), 18-26.
- Alam, B., & Kumar, A. (2013). Fault Tolerance Issues in Real Time Systems with Energy Minimization. *International Journal of Information and Computation Technology*, 1001-1008.
- Alam, M., & Varshney, A. K. (2016). A New Approach of Dynamic Load Balancing Scheduling Algorithm for Homogeneous Multiprocessor System. *International Journal of Applied Evolutionary Computation (IJAEC)*, 7(2), 61-75.
- AlEnawy, T. A., & Aydin, H. (2005, March). Energy-aware task allocation for rate monotonic scheduling. In *11th IEEE Real Time and Embedded Technology and Applications Symposium* (pp. 213-223). IEEE.
- Bautista, D., Sahuquillo, J., Hassan, H., Petit, S., & Duato, J. (2008, April). A simple power-aware scheduling for multicore systems when running real-time applications. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (pp. 1-7). IEEE.
- Berl, A., Gelenbe, E., Di Girolamo, M., Giuliani, G., De Meer, H., Dang, M. Q., & Pentikousis, K. (2010). Energy-efficient cloud computing. *The computer journal*, 53(7), 1045-1051.
- Bokhari, M. U., Alam, M., & Hasan, F. (2016). Performance analysis of dynamic load balancing algorithm for multiprocessor interconnection network. *Perspectives in Science*, 8, 564-566.
- Buttazzo, G. (2011). *Hard real-time computing systems: predictable scheduling algorithms and applications* (Vol. 24). Springer Science & Business Media.
- Carr, J. (2014). An introduction to genetic algorithms. See: karczmarczyk.users.greyc.fr/TEACH/IAD/GenDoc/carrGenet.pdf.

- Chandak, A. V., Sahoo, B., & Turuk, A. K. (2011). Heuristic task allocation strategies for computational grid. *Int. J. Advanced Networking and Applications*. Volume: 02, Issue: 05, Pages: 804-810
- Chang, R. S., Chang, J. S., & Lin, P. S. (2009). An ant algorithm for balanced job scheduling in grids. *Future Generation Computer Systems*, 25(1), 20-27.
- Chen, J. J., & Kuo, T. W. (2005). Energy-efficient scheduling of periodic real-time tasks over homogeneous multiprocessors. *PARC (September 2005)*, 30-35.
- Chen, J. J., & Thiele, L. (2008, August). Expected system energy consumption minimization in leakage-aware dvs systems. In *Proceedings of the 2008 international symposium on Low Power Electronics & Design* (pp. 315-320). ACM.
- Chen, J. J., Yang, C. Y., Lu, H. I., & Kuo, T. W. (2008, April). Approximation algorithms for multiprocessor energy-efficient scheduling of periodic real-time tasks with uncertain task execution time. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE* (pp. 13-23). IEEE.
- Coffman Jr, E. C., Garey, M. R., & Johnson, D. S. (1996). Approximation algorithms for bin packing: A survey. *Approximation Algorithms for NP-Hard Problems*, 46-93
- D. G. Harkut ,& Anuj M. Agrawal (2014). A survey of Comparison of Different Task Scheduling Algorithms in RTOS, *International Journal of Advanced Research in Computer Science and Software Engineering* Volume 4, Issue 7, July 2014 ISSN: 2277 128X, 1237-1240
- Dai, Y. S., & Wang, X. L. (2006). Optimal resource allocation on grid systems for maximizing service reliability using a genetic algorithm. *Reliability Engineering & System Safety*, 91(9), 1071-1082.
- Dalgaard, Peter (2008). *Introductory Statistics with R*. Springer Science & Business Media. (pp. 99-100). ISBN 978-0-387-79053-4.
- Davis, R. I. (2014). A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Review*, 11(1), 8-19.

- Devi, M. R., & Anju, A. (2014). Multiprocessor scheduling of dependent tasks to minimize makespan and reliability cost using NSGA-II. *arXiv preprint arXiv:1404.2739*.
- Dick, R. P., Rhodes, D. L., & Wolf, W. (1998, March). TGFF: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign* (pp. 97-101). IEEE Computer Society.
- Eidenbenz, R., & Locher, T. (2016). Task Allocation for Distributed Stream Processing. *arXiv preprint arXiv:1601.06060*.
- Falsafi, B., & Vijaykumar, T. N. (Eds.). (2004). *Power-Aware Computer Systems: Third International Workshop, PACS 2003, San Diego, CA, USA, December 1, 2003, Revised Papers* (Vol. 3164). Springer.
- Fayad, C., Garibaldi, J. M., & Ouelhadj, D. (2007, July). Fuzzy Grid Scheduling Using Tabu Search. In *FUZZ-IEEE* (pp. 1-6).
- Fidanova, S. (2006, October). Simulated annealing for grid scheduling problem. In *IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing (JVA'06)* (pp. 41-45). IEEE.
- Fister Jr, I., Yang, X. S., Fister, D., & Fister, I. (2014). Cuckoo search: a brief literature review. In *Cuckoo search and firefly algorithm* (pp. 49-62). Springer International Publishing.
- Foster, I., & Kesselman, C. (Eds.). (2003). *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier.
- Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3), 200-222.
- Gao, Y., Rong, H., & Huang, J. Z. (2005). Adaptive grid job scheduling with genetic algorithms. *Future Generation Computer Systems*, 21(1), 151-161.
- Golub, M., & Kasapovic, S. (2002, January). Scheduling multiprocessor tasks with genetic algorithms. In *APPLIED INFORMATICS-PROCEEDINGS-* (No. 3, pp. 273-278).
- Goraczko, M., Liu, J., Lymberopoulos, D., Matic, S., Priyantha, B., & Zhao, F. (2008, June). Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *Proceedings of the 45th annual design automation*

- conference (pp. 191-196). ACM.
- Halang, W. A., & Stoyenko, A. D. (1994). *Real Time Computing*. Springer-Verlag New York, Inc..
- Haupt, R. L., & Haupt, S. E. (2004). *Practical genetic algorithms*. John Wiley & Sons.
- Hua, S., Qu, G., & Bhattacharyya, S. S. (2003, June). Energy reduction techniques for multimedia applications with tolerance to deadline misses. In *Proceedings of the 40th annual Design Automation Conference* (pp. 131-136). ACM.
- Huang, L., & Xu, Q. (2010, March). Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint. In *Proceedings of the conference on Design, automation and test in Europe* (pp. 1584-1589). European Design and Automation Association.
- Isovic, D., & Fohler, G. (2000). Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE* (pp. 207-216). IEEE.
- Jejurikar, R. (2005, July). Energy aware non-preemptive scheduling for hard real-time systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)* (pp. 21-30). IEEE.
- Jejurikar, R., & Gupta, R. (2006). Energy-aware task scheduling with task synchronization for embedded real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6), 1024-1037.
- Jin, Y., Jin, J., Gluhak, A., Moessner, K., & Palaniswami, M. (2012). An intelligent task allocation scheme for multihop wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(3), 444-451.
- Jin, Y., Vural, S., Gluhak, A., & Moessner, K. (2013). Dynamic task allocation in multi-hop multimedia wireless sensor networks with low mobility. *Sensors*, 13(10), 13998-14028.
- Juang, P., Wu, Q., Peh, L. S., Martonosi, M., & Clark, D. W. (2005, August). Coordinated, distributed, formal energy management of chip multiprocessors. In *ISLPED'05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.* (pp. 127-130). IEEE.
- Kadayif, I., Kandemir, M., & Sezer, U. (2002, June). An integer linear programming based approach for parallelizing applications in

- on-chip multiprocessors. In *Proceedings of the 39th annual Design Automation Conference* (pp. 703-706). ACM.
- Kamiura, J., Hiroyasu, T., Miki, M., & Watanabe, S. (2002, August). MOGADES: multi-objective genetic algorithm with distributed environment scheme. In *Second international workshop on Intelligent systems design and application* (pp. 143-148). Dynamic Publishers, Inc.
- Kang, J., & Ranka, S. (2010). Dynamic slack allocation algorithms for energy minimization on parallel machines. *Journal of Parallel and Distributed Computing*, 70(5), 417-430.
- Kargahi, M., & Movaghar, A. (2008). Stochastic DVS-based dynamic power management for soft real-time systems. *Microprocessors and Microsystems*, 32(3), 121-144.
- Kousalya, K., & Balasubramanie, P. (2009). To Improve Ant Algorithm's Grid Scheduling Using Local Search. *International Journal of Intelligent Information Technology Application*, 2(2).
- Kousalya.K and Balasubramanie.P, (December 2008), Ant Algorithm for Grid Scheduling Powered by Local Search, Int. J. Open Problems Compt. Math., Vol. 1, No. 3.
- Kumar Tripathi, A., Kumer Sarker, B., Kumar, N., & Vidyarthi, D. P. (2000). A GA based multiple task allocation considering load. *International Journal of High Speed Computing*, 11(04), 203-214.
- Kumar, G. S. A., Manimaran, G., & Wang, Z. (2009). Energy-aware scheduling with probabilistic deadline constraints in wireless networks. *Ad Hoc Networks*, 7(7), 1400-1413.
- Lee, C., Potkonjak, M., & Mangione-Smith, W. H. (1997, December). MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture* (pp. 330-335). IEEE Computer Society.
- Lee, Y. H., & Chen, C. (2003). A modified genetic algorithm for task scheduling in multiprocessor systems. In *the 9th workshop on compiler techniques for high-performance computing*.
- Lin, M., & Ng, S. M. (2005). A genetic algorithm for energy aware task scheduling in heterogeneous systems. *Parallel processing letters*, 15(04), 439-449.
- Lin, M., & Ng, S. M. (2005). Energy Aware Scheduling for

- Heterogeneous Real-Time Embedded Systems Using Genetics Algorithms. In *New Horizons of Parallel and Distributed Computing* (pp. 113-128). Springer US.
- Liu, C. (2013). *Efficient Design, Analysis, and Implementation of Complex Multiprocessor Real-Time Systems* (Doctoral dissertation, University of North Carolina at Chapel Hill).
- Liu, F., & Tsui, C. Y. (2006, May). Energy-aware optimal workload allocation among the battery-powered devices to maximize the co-operation life time. In *2006 IEEE International Symposium on Circuits and Systems* (pp. 4-pp). IEEE.
- Liu, L., Yang, Y., Li, L., & Shi, W. (2006, December). Using ant colony optimization for superscheduling in computational grid. In *2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06)* (pp. 539-545). IEEE.
- Mahapatra, R. N., & Zhao, W. (2005). An energy-efficient slack distribution technique for multimode distributed real-time embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(7), 650-662.
- Malakooti, B., Sheikh, S., Al-Najjar, C., & Kim, H. (2013). Multi-objective energy aware multiprocessor scheduling using bat intelligence. *Journal of Intelligent Manufacturing*, 24(4), 805-819.
- Miao, L., Qi, Y., Hou, D., Dai, Y. H., & Shi, Y. (2008, October). A multi-objective hybrid genetic algorithm for energy saving task scheduling in CMP system. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on* (pp. 197-201). IEEE.
- Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.
- Mohamad, A., Zain, A. M., Bazin, N. E. N., & Udin, A. (2013). Cuckoo Search Algorithm for Optimization Problems-A Literature Review. In *Applied Mechanics and Materials* (Vol. 421, pp. 502-506). Trans Tech Publications.
- Mohammadi, A. (2009). *Scheduling algorithms for real-time systems*, (Doctoral dissertation, Queen's University Kingston, Ontario, Canada). Retrieved from <http://www.collectionscanada.gc.ca/obj/thesescanada/vol2/OKQ/TC-OKQ-1787.pdf>
- Mohammadi, A., & Akl, S. G. (2005). Scheduling algorithms for real-

- time systems. *School of Computing, Queens University*. Retrieved from <http://research.cs.queensu.ca/home/akl/techreports/scheduling.pdf>
- Mottaghi, M. H., & Zarandi, H. R. (2014). DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors. *Microprocessors and Microsystems*, 38(1), 88-97.
- Nedunchelian, R., Koushik, K., Meiyappan, N., & Raghu, V. (2006). Dynamic Task Scheduling Using Parallel Genetic Algorithms For Heterogeneous Distributed Computing. In *GCA* (pp. 82-88).
- Nelis, V., & Goossens, J. (2009, August). Mora: An energy-aware slack reclamation scheme for scheduling sporadic real-time tasks upon multiprocessor platforms. In *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* (pp. 210-215). IEEE.
- Obitko, M. (2005). *Genetic Algorithms*. Retrieved from <http://www.obitko.com/tutorials/genetic-algorithms/>
- Oh, J., & Wu, C. (2004). Genetic-algorithm-based real-time task scheduling with multiple goals. *Journal of systems and software*, 71(3), 245-258.
- Ouyang, A., Pan, G., Yue, G., & Du, J. (2014). Chaotic cuckoo search algorithm for high-dimensional functions. *Journal of Computers*, 9(5), 1282-1290.
- Ouyang, X., Zhou, Y., Luo, Q., & Chen, H. (2013). A novel discrete cuckoo search algorithm for spherical traveling salesman problem. *Applied mathematics & information sciences*, 7(2), 777.
- Page, A. J., & Naughton, T. J. (2005, April). Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. In *19th IEEE international parallel and distributed processing symposium* (pp. 189a-189a). IEEE.
- Page, A. J., Keane, T. M., & Naughton, T. J. (2010). Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. *Journal of parallel and distributed computing*, 70(7), 758-766.
- Paulinas, M., & Ušinskas, A. (2015). A survey of genetic algorithms applications for image enhancement and segmentation. *Information Technology and control*, 36(3).

- Pencheva, T., Angelova, M., Atanassova, V., & Roeva, O. (2015). InterCriteria analysis of genetic algorithm parameters in parameter identification. *Notes Intuitionistic Fuzzy Sets*, 21(2), 99-110.
- Prosperi, F., Bambagini, M., Buttazzo, G., Marinoni, M., & Franchino, G. (2012, September). Energy-aware algorithms for tasks and bandwidth co-allocation under real-time and redundancy constraints. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)* (pp. 1-8). IEEE.
- Rodrigues, D., Pereira, L. A., Almeida, T. N. S., Papa, J. P., Souza, A. N., Ramos, C. C., & Yang, X. S. (2013, May). BCS: A Binary Cuckoo Search algorithm for feature selection. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)* (pp. 465-468). IEEE.
- Roeva, O., Fidanova, S., & Paprzycki, M. (2013, September). Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. In *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on* (pp. 371-376). IEEE.
- Rouhifar, M., & Ravanmehr, R. (2015). A Survey on Scheduling Approaches for Hard Real-Time Systems. *International Journal of Computer Applications*, 131(17), 41-48.
- Samal, A. K., Mall, R., & Tripathy, C. (2014). Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm. *Swarm and Evolutionary Computation*, 14, 92-105.
- Sastry, K., Goldberg, D. E., & Kendall, G. (2014). Genetic algorithms. In *Search methodologies* (pp. 93-117). Springer US. Retrieved from: http://webpace.ulbsibiu.ro/adrian.florea/html/Planificari/EvolutionaryComputing/Course_4/pdf/introsch4.pdf
- Shin, D., & Kim, J. (2003, August). Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In *Proceedings of the 2003 international symposium on Low power electronics and design* (pp. 408-413). ACM.
- Srinivasan, B., Pather, S., Hill, R., Ansari, F., & Niehaus, D. (1998, June). A firm real-time system implementation using commercial off-the-shelf hardware and free software. In *Real-Time Technology and Applications Symposium, 1998*.

- Proceedings. Fourth IEEE* (pp. 112-119). IEEE.
- Swaminathan, V., & Chakrabarty, K. (2001). Real-time task scheduling for energy-aware embedded systems. *Journal of the Franklin Institute*, 338(6), 729-750.
- Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems*. Prentice-Hall.
- Tavares, E., Maciel, P., Silva, B., & Oliveira, M. N. (2008). Hard real-time tasks' scheduling considering voltage scaling, precedence and exclusion relations. *Information Processing Letters*, 108(2), 50-59.
- Tindell, K. W., Burns, A., & Wellings, A. J. (1992). Allocating hard real-time tasks: an NP-hard problem made easy. *Real-Time Systems*, 4(2), 145-165.
- Tsang, P. W., & Au, A. T. (1996, November). A genetic algorithm for projective invariant object recognition. In *TENCON'96. Proceedings., 1996 IEEE TENCON. Digital Signal Processing Applications* (Vol. 1, pp. 58-63). IEEE.
- Valian, E., Mohanna, S., & Tavakoli, S. (2011). Improved cuckoo search algorithm for feedforward neural network training. *International Journal of Artificial Intelligence & Applications*, 2(3), 36-43.
- Van Steen, M. (2003). Distributed systems principles and paradigms. *Network*, 3, 26.
- Walton, S., Hassan, O., Morgan, K., & Brown, M. R. (2011). Modified cuckoo search: a new gradient free optimisation algorithm. *Chaos, Solitons & Fractals*, 44(9), 710-718.
- Wang, W., & Mishra, P. (2010, June). PreDVS: Preemptive dynamic voltage scaling for real-time systems using approximation scheme. In *Proceedings of the 47th Design Automation Conference* (pp. 705-710). ACM.
- Wu, A. S., Yu, H., Jin, S., Lin, K. C., & Schiavone, G. (2004). An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on parallel and distributed systems*, 15(9), 824-834.
- Khafa, F., Alba, E., Dorronsoro, B., Duran, B., & Abraham, A. (2008). Efficient batch job scheduling in grids using cellular memetic algorithms. In *Metaheuristics for Scheduling in Distributed Computing Environments* (pp. 273-299). Springer Berlin Heidelberg.

- Xian, C., Lu, Y. H., & Li, Z. (2007, June). Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *Proceedings of the 44th annual Design Automation Conference* (pp. 664-669). ACM.
- Yang, X. S., & Deb, S. (2009). Cuckoo search via Lévy Flights. *World Congress on Nature & Biologically Inspired Computing. IEEE*.
- Yang, X. S., & Deb, S. (2010). Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, 1(4), 330-343.
- Yao, F., Demers, A., & Shenker, S. (1995, October). A scheduling model for reduced CPU energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on* (pp. 374-382). IEEE.
- Yasar, M. (2016). Optimization of Reservoir Operation Using Cuckoo Search Algorithm: Example of Adiguzel Dam, Denizli, Turkey. *Mathematical Problems in Engineering*, 2016.
- Z. Lu, Y. Zhang, M. Stan, J. Lach, & K. Skadron (2006). Procrastinating voltage scheduling with discrete frequency sets. In *Design, Automation and Test in Europe (DATE)*, (pp 456-461).
- Zhang, Y., Hu, X. S., & Chen, D. Z. (2002, June). Task scheduling and voltage selection for energy minimization. In *Proceedings of the 39th annual Design Automation Conference* (pp. 183-188). ACM.
- Zhang, Y., Lu, Z., Lach, J., Skadron, K., & Stan, M. R. (2005, June). Optimal procrastinating voltage scheduling for hard real-time systems. In *Proceedings of the 42nd annual Design Automation Conference* (pp. 905-908). ACM.
- Zheng, H., Zhou, Y., He, S., & Ouyang, X. (2012). A discrete cuckoo search algorithm for solving knapsack problems, *Advances in Information Sciences and Service Sciences* (PP. 331-339)
- Zhu, D., Melhem, R., & Childers, B. R. (2003). Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(7), 686-700.
- Zhu, J., Li, J., & Gao, H. (2007). Energy Efficient and QoS-Aware Tasks Allocation for Sensor Networks. *Journal of Digital*

Information Management,5(2), 88.

- Zhu, Y., & Mueller, F. (2004, May). Feedback EDF scheduling exploiting dynamic voltage scaling. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE* (pp. 84-93). IEEE.
- Zomaya, A. Y., Ward, C., & Macey, B. (1999). Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed systems*, 10(8), 795-812.

١٦ يناير ٢٠١٧ م

سعادة الدكتور / محمد رضا قادر الموقر
عميد عمادة الدراسات العليا والبحث العلمي

تحية طيبة وبعد ،،،

بناءً على المادة (٦١) من الفصل الحادي عشر من نظام الدراسات العليا بجامعة البحرين وملاحقه ، يسرني أن أرفق لكم نسخة من أطروحة الماجستير للطالبة نور عدنان عواد (الرقم الجامعي ٢٠٠٢٧٥٦٤) والتي قد أنهت بنجاح متطلبات التخرج من برنامج الماجستير في تقنية المعلومات في الفصل الدراسي الثاني من العام الجامعي ٢٠١٦/٢٠١٧ م.

هذا، وتفضلوا بقبول خالص التحية والتقدير،،،

الدكتور هشام محمد العمال
عميد كلية تقنية المعلومات



الملخص

أصبح موضوع كفاءة استهلاك الطاقة واحدا من أهم القضايا في تصميم مختلف الأنظمة. يتأثر استهلاك الطاقة في الأنظمة المتعددة المعالجات بشكل كبير بالطريقة المتبعة لجدولة المهام عليه. وقد استحوذت مشكلة تصميم أنظمة أقل استهلاكاً للطاقة على اهتمام الأوساط البحثية في مجال علوم الحاسوب في الآونة الأخيرة، حيث تم التعامل معها كهدف للتصميم في الأنظمة المحدودة الطاقة والأنظمة المتنقلة، خاصة لأنظمة الوقت الحقيقي.

تتناول هذه الأطروحة مشكلة جدولة المهام في الوقت الحقيقي باستخدام التغيير الديناميكي لسرعة المعالجات لمنظومة معالجات متعددة. حيث نتناول مشكلة جدولة المهام في الوقت الحقيقي على اعتبارها مشكلة من مشاكل التحسين التي تخضع للعديد من القيود المتعلقة بطبيعة النظام. قد تم اقتراح وتصميم العديد من الخوارزميات التي تسعى لإيجاد أفضل أو شبه أفضل النتائج للمشكلة المتناولة. اعتمدنا في هذه الأطروحة على تكييف بعض الخوارزميات المستوحاة من الطبيعة مثل الخوارزمية الجينية (GA) وخوارزمية بحث الوقواق (CS) بما يتناسب لحل المشكلة الأساسية لهذا البحث. تم استخدام تحليل حساسية العوامل على كفاءة الخوارزميات المقترحة لتحديد أنسب التراكيب لتحقيق أفضل النتائج من حيث جودة الحل وكفاءة وقت المعالجة والتنفيذ لكل من الخوارزميات التي تم تكييفها في هذا البحث. تم اقتراح تغييرات متعددة في تصميم عوامل العبور والتحول في الخوارزمية الجينية، و تم اقتراح طريقة لتحديد القيم المناسبة لعوامل خوارزمية بحث الوقواق. كما واقترحنا تصميمًا معدلاً لخوارزمية بحث الوقواق وفيه يتم أخذ بعض الحلول غير المجدية في الاعتبار لإيجاد حل مجد وفعال على أن يضاف لها غرامة بطريقة يتم تحديدها. حققت الخوارزميات المقترحة توفيراً في الطاقة المستهلكة لما يصل إلى 89% في أفضل الحالات، مع قدرتها على تلبية جميع القيود المرتبطة بنظام الوقت الحقيقي بنجاح. وقد توصلت الخوارزمية الجينية GA و خوارزمية بحث الوقواق CS في معظم الحالات إلى نتائج مماثلة من حيث جودة الحل. بشكل عام فقد أظهرت نتائج المحاكاة على منظومة متعددة المعالجات مكونة من خمسة معالجات تفوق الخوارزمية الجينية على خوارزمية الوقواق من حيث وقت التنفيذ بنسبة تصل إلى 58% من حيث وقت التنفيذ بينما كانت أفضل بنسبة 50% من خوارزمية بحث الوقواق مع اضافة غرامة للحلول الغير مجدية.

جامعة البحرين

كلية تقنية المعلومات



رقيب جدولة الطاقة لمنظومة معالجات متعددة بالزمن الحقيقي

مخطط بحث لاستكمال متطلبات الحصول على درجة الماجستير
في تقنية المعلومات

إعداد

نور عدنان عواد

الرقم الجامعي: 20027564

إشراف

الدكتور أمجد محمود

(أستاذ مشارك)

جامعة البحرين

مشرف مساعد

الدكتور سلمان خان

(أستاذ مساعد)

جامعة البحرين

مملكة البحرين

أكتوبر 2016