On Static Scheduling of Tasks in Real Time Multiprocessor Systems: An Improved GA-Based Approach

Mohammad Ababneh, Salama Hassan, and Sulieman Bani-Ahmad
Prince Abdullah Bin Ghazi of Information Technology, Al-Balqa Applied University, Jordan

Abstract: *Task execution Deadline Time (DL) in real-time systems is a critical constraint. Every task should have a Maximum Computational Time (MCT) that is needed before reaching a given DL time. Scheduling jobs in real-time systems is thus a nondeterministic polynomial NP problem. Three algorithms can be found in literature to solve these problems in a multi processor environment; are the Earliest Deadline First (EDF), Genetic Algorithms (GA), Priority Genetic Algorithms (PGA). In this research, the PGA is introduced and experimentally evaluated against already proposed algorithms in literature. It works just like the GA algorithm introduced in Abraham et al. [1]. However, we do not only consider the DL in sorting the tasks in the first population, but rather, we also include the MCT of individuals in the population to define the priority level of these tasks. We have found that the proposed algorithm has a better average total system utilization, total system tasks visibility compared with Genetic (G) and EDF algorithms. We have also found that this improvement becomes more and more effective with the increase of problem size.*

Keywords: *Task scheduling, multiprocessor systems, GA.*

Received December 10, 2012; accepted June 2, 2013; published online March 13, 2014

1. Introduction

Real time systems do not only care about correct results of jobs. In fact, DeadLine time (DL) is a critical constraint to any task. Every task should have a Maximum Computational Time (MCT) that is needed before reaching that given DL [10]. Scheduling jobs in realtime systems is thus an NP-problem [22].

One of the static multi processor scheduling problems is processor number, where multi processor system that includes heterogeneous set of processors with different specifications. Such systems can process the task or job with different execution time [10]. Considering dependency between these tasks and minimizing the probability of having deadlocks, are all objectives of scheduling that can have Earliest Deadline First (EDF) and Shortest Computation Time First (SCTF) met [1].

Genetic Algorithms (GAs) or evolutionary algorithms are random search techniques that are based on the evolutionary ideas of natural selection. GAs can be applied on the problem of scheduling in real time multi processor tasks and enhance the GA operations to reach better optimal solutions to this problem.

2. The Reflective Process

2.1. CPU Tasks Scheduling

Multi processor systems have evolved from the single-processor systems to support multiprogramming which aims to keep some processes running at all times.

In single-processor system, only one process (task) allocates the CPU and the other processes (tasks) should wait until the currently running process finishes the Computational Time (CT) that it needs. Although, sometimes this process may need to do some I/O operations, the CPU will be idle as the process does the I/O operation, this should significantly reduce the CPU utilization and reduce the served process in time period (that is the throughput of the system) [8].

The main goal of multiprogramming is to protect CPU from processes which reserve CPU all the time. In such systems, all processes should be loaded into the main memory, if the CPU is occupied by another process, the new-fresh-process shall wait for the operating system to take the CPU from the running process and give it to the waiting processes that wait in the ready queue [8, 24].

CPU scheduler is part of the operating system and is responsible for managing the allocation of the CPU among active processes [23].

The problem of CPU scheduling belongs to the NP-Hard problems. In fact, the scheduling algorithm can significantly affect the CPU utilization [7], therefore, many scheduling algorithms have been proposed in literature.

2.2. Real Time Systems

Real-time systems are applied and currently used in many areas of our every-day life such as defense, scientific research, transportation, management, network communication, meetings through a web video and/or audio etc., [12, 14].

Processes scheduling to resources in real time systems face many challenges. The most important of these challenges are:

1. *Large Search Spaces*: Generally, the scheduling means assigning N tasks to one of M resources in specific order. This means that there are possibilities to make orderings of the N tasks on M resources. Even if there is only one resource in the system there will be different orderings of the N tasks [19].
2. *Dynamically Changing Problems*: Almost all tasks running in a real time system constantly receive updates. These tasks vary in their level of priority. This adds complexity to the problem of scheduling in hand [19].
3. *A Variety of Constraints*: There are two types of constraints in scheduling problems. The first type is called "hard" and the second is called 'soft'. Hard constraints are the ones which should be satisfied for the schedule to be considered legal, while soft constraints are essentially preferable [19].

2.3. Multi Processor Real Time Scheduling

In some applications, using a single-processor is not enough to work typically as it should, so many problems such as long response time, low throughput etc., will appear in such systems.

Sometimes, there is a need to have multi processor system in which we distribute the computational load efficiently among the available CPU's. For that, it is necessary to divide the entire task into subtasks and to properly arrange the order of the execution of these subtasks [15].

Multi processor systems provide suitable environment and are more powerful to run real time applications than uniprocessor systems. That's why scheduling in multi processor system has been an active field of research.

In such systems, every task in the multi processor system works in a way, that makes it look like a uniprocessor scheduling, centralized multiprocessor scheduling and distributed scheduling [12].

There are complicated load calculations in multi processor real-time scheduling. The scheduling in multiprocessor system is not only to sort tasks, but also to allocate them to processors. This means that scheduling algorithms in multi processor systems are much more complicated than those of uniprocessor [12].

There are two strategies in multi processor scheduling: Global scheme, and division scheme. In global scheduling scheme, real-time tasks are run on different processors every time. Tasks can be preempted before their implementation and be transferred among different processors. In division scheduling, a task is run on the same processor in different times. All tasks are assigned to processors by task allocation algorithm in advance [16].

2.4. Dynamic vs. Static Multi Processor Scheduling

There is a difference between dynamic and static multi processor scheduling. While dynamic scheduling deals with jobs when they arrive at the scheduler and can deal with changing the numbers of processors, in static scheduling, however, all necessary information about the jobs and processors should be known before scheduler runs an algorithm. For example, the scheduler should know the Running times (RN). This means that after some fixed time, the algorithm will be re-executed. Also, in static scheduling, the number of the processors available in the system is assumed to be known as well as the number of tasks to be scheduled [9].

2.5. Problem Specifications

The most important thing in real time CPU scheduling is to make response time to the given process, which requests a CPU to be as short as possible.

Real time CPU scheduler should serve the processes before their DLs are reached. Also, it should be clear to the CPU scheduler that serving the process after its DL is expired will be meaningless because it was out of time.

Every process has mainly three characteristics. The first is the process Arrival Time (AT): That is, the time when the process entered the ready queue. The second is the process CT: That is, the fixed period of time that process needs to be executed on resource. The third is the process DL time: That the process should finish the CT before reaching that DL [8]. Figure 1 describes the process characteristics.

According to the above three characteristics, any process in ready queue should comply with the following relations $0 < AT < DL$ and $CT < DL - AT$.

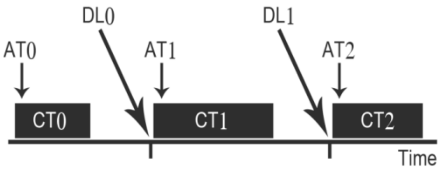


Figure 1. Characteristics of processes.

3. Literature Review

In the last few years, a number of papers were published that cover the real time system scheduling using GA.

Agarwal *et al.* [2] a group of researchers developed a new technique, based on a GA depends on the principles of evolution found in nature for finding an optimal solution. GA is based on three operators: Natural selection, crossover and mutation. In this technique GA use static scheduling to find optimal solution which proved to be efficient to find optimal solution more than Heterogeneous Earliest Finish Time (HEFT) with same length of problem size focusing on the quality of solution and effect of mutation probability on the performance of GA.

In [1] a group of researchers proposed GA to generate dynamic real time tasks scheduling system improve visibility than EDF and SCTF. The purpose of this scheduling algorithm is to enhance the utilization of the processors as it depends on the earliest DL first to sort the tasks first population. In the researchers approach, the GA consist of the following steps:

1. Generate a task queue.
2. Sort the tasks in the increasing order of their DLs.
3. Select a suitable number of tasks for a fixed chromosome size.
4. Generate chromosomes for the population.
5. Sort the genes in each chromosome based on DL.
6. Determine the fitness value of each chromosome in the population.
7. Sort the chromosomes within the population depending on fitness value.
8. Apply GA operators for a number of iterations.
9. Choose the best chromosome.

A better performance is obtained by using GA with the heuristic. The percentage of tasks that are feasible is 95 percent and above Abraham *et al.* [1].

ManChon *et al.* [18] a group of researchers proposed new Real-Time system scheduler based on Genetic Algorithm (GART) by applying DP-Wrap technique trying to reduce the overhead due to preemption and migration by rearranging the schedule so as to increase the duration between preemption , this approach answer two questions. First, what is the best heuristic? Second, is the same heuristic best for all real-time systems? The experimental results explained that this algorithms can produce the best heuristic for all the systems considered.

Rashtbar *et al.* [21] a group of researchers developed new Hybrid technique depends on GA based on neighborhood search and tabu search for performing Task Scheduling (HGTS). This technique focuses on the results quality and execution time of algorithm, it produce of appropriate task schedule by spending less execution time since, there should be a balance between solution space and execution time of algorithm.

Dandass [5] has developed a new hybrid technique depending on List Scheduling (LS) with a GA for constructing non-preemptive schedules for soft real-time parallel applications. This technique consist ot two phases: First one depends on hybrid GA and LS approach is used to construct a preliminary schedule based on a fixed estimate of task execution times. In second phase the preliminary schedule consummated in the first phase is converted into a stochastic schedule by using PDF operations wich improve shorter schedules than two popular LS approaches for a majority of sample problems.

Ceyda and Ercan [3] a group of researchers have applied a GA from multi-layer multiprocessor task scheduling and introduce a new crossover operator. Where this new crossover compared to PMX crossover which is proofed to be the best performing crossover technique.

Zhu *et al.* [25] a group of researchers have put forward a task scheduling algorithm based on self-adjusting GA and grid computing to schedul heterogeneous collections of remote is essentially to distribute N interdependent tasks to resources. Generating a fitness function throught weighted least connection algorithm then generating a new population of individuals through genetic operation (reproduction, crossover, mutation) where this approach reduce task complete time. The result here is more ideal than other scheduling algorithms.

Pooranian *et al.* [20] a group of researchers have developed a novel task scheduling based on hybrid GA and GELS algorithm. This technique have solved grid scheduling problem and minimize missed tasks. In this approach every chromosome represents visible solution, and move (pick) solution after GA operation that better than current solution using purpose function (fitness function) and some of advantage of GLES algorithm in random search. This algorithm proved that can decreases the number of missed tasks more than other algorithms.

Ilavarasan and Thambidurai [11] a group of researchers have proposed a new task scheduling for distributed heterogeneous computing system namely GATS. This tasks provide optimal results for acyclic graph, where the performance of this algorithm (schedule length, speedup and efficiency) compared with existing algorithms such as CPOP, HEFT and PSGA where the final result show that the GATS substantially outperforms these algorithms.

Man and Sai [17] a group of researchers have proposed a new static scheduling approach for heterogeneous systems using GA. That depends on tasks executing on a number of heterogeneous processors to reduce the energy consumption. The simulation result show that this approach reduce the energy consumption more than other three algorithm namely EDF, longest-time-first and simulated-anealing by an estimated 20% to 90% under different system configurations.

Daoud and Kharma [6] a group of researchers have proposed a new approach namely (GS) which uses a customized GA. This approach is to produce high-quality tasks schedules for Heterogeneous Distributed Computing Systems (HeDCSs) and compare the terms of average schedule length, speedup and efficiency in the HEFT algorithm and the DLS algorithm where GS significantly outperforms the traditional scheduling algorithm.

Cheng and Huang [4] present a GA-based approach with a feasible energy function that generate good-quality schedules due the crossover and mutation operators.

Kazem *et al.* [13] a group of researchers have presented a GA for scheduling the independent tasks in isochronal soft real-time systems. The suggested algorithm is a TUF based scheduling algorithm that its objective is maximizing the sum of utilities attained by jobs. Different experiments indicate that the proposed GA has not only high stability, but also high convergence. Experiments also show that the proposed GA produces schedules that the total utility accrued by the system is high.

4. Implementation of Previous Attempts to Enhancing Real Time Multi Processor Using Genetic Algorithm

As we mentioned before, many of researches on multi processors tasks scheduling using GA were initiated.

Next, we will focus on the heuristics algorithm such as EDF and we will compare the results with GA. The feedback that we got will produce a new enhancement for the GA, the complexity of system can be measured by the CPUs number.

Some systems may contain few of CPUs (small systems) and heterogeneous CPUs (large systems) where some algorithms can be applied efficiently on small systems and can't deal with large systems effectively. The complexity of system depends on the number of CPUs but, it depends on the size of the problem as well (number of tasks).

We will discuss the utilization and visibility of CPUs that can be found through EDF and GA to find optimal solution.

The GA has the iteration number to make an enhancement for the current optimal solution when applying GA operations, but this process will requir longer time. The relation between the GA iteration and execution time of the algorithm is a direct proportion, it means when the iteration number for GA is increased, we will get better optimal solution, but the algorithm will take more time to be executed. However, keep in mind that the time is a very important factor in the real time system.

4.1. Our Simulator

In this phase, a simulator is built using Microsoft Visual Basic.NET (MVB.NET) and external toolkit called devexpress. This simulator is used to describe the difference between the EDF and GA and to represent the parameter as Figures 4-1 (e.g. total system utilization) where we can see the effects of changing some of GA operation on the optimal solution. This simulator got most functions in GA. The input to our simulator are: Number of processor, number of processes (tasks) where it will be scheduled on processor. Simulator takes the RN to generate random tasks which its AT should be less than or equal to the RN. The iteration number will repeat the genetic operation N times. This simulator can take two techniques to sort the first populations. The crossover type (an input to our simulator) describes two types of crossover while.

Our simulator supports three kinds of replacement methods. The child chromosomes that are generated after applying GA operations will be replaced by their parents to proof the following factors (we will discuss them later):

1. Total fitness function is best.
2. Each chromosome fitness function is best.
3. Ignore fitness, just replace.

The simulator supports the following selection methods:

1. Randomly.
2. Incrementally.
3. Random and incremental.

This simulator can change the fitness function to evaluate the chromosome, where there are three types of fitness function:

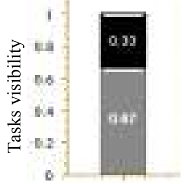
1. Chromosome visibility.
2. Chromosome utilization and visibility.
3. Chromosome utilization.

4.2. Earliest Deadline First vs. Genetic Algorithms

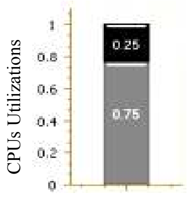
In this section, we will discuss how the optimal solution will be affected when some of GA operations are changed in the same problem. Suppose that we have the following problem: 18 tasks, 3 CPUs and RN equals to 60ns, we will use GA and EDF algorithms to find the best algorithm.

4.2.1. Earliest Deadline First Algorithm

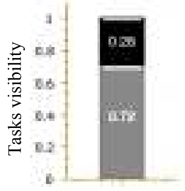
Figure 2-a shows the result that came using EDF algorithm with system visibility 0.67, this system executes only 67% tasks.



a) Total system tasks visibility using EDF algorithm.



b) Total system utilization using EDF algorithm.



c) Total system visibility using GA.

Figure 2. EDF algorithm results.

We got this result because all tasks are arranged in task queue by their DL, as the task with the lowest DL will be executed first on CPU. However, this algorithm ignores AT and MCT for each task.

For example, assume that we have two tasks to be scheduled, T1 and T2. These tasks have DLs equals to 7 and 8, MCTs equals to 1 and 5, ATs equals to 3 and 1 and the current time in CPU is zero. In this case the CPU starts with task T1 because its DL here is less than T2, the CPU will stay idle until CT equals to 3. Task T1 will be scheduled and will finish at time CT + MCT. After T1 finishes, the CT equals to 4, then the CPU will try to execute T2 but time CT + MCT is, obviously, greater than the DL of T2. As a result, we end up ignoring T2 and that will decrease the total system visibility and utilization because CPU will stay idle until CT reaches the AT of T1.

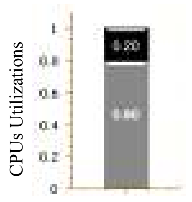
This will effect the CPU utilization as it's value will become 75% as shown in Figure 2-b.

4.2.2. Genetic Algorithm

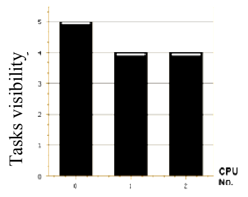
If we are trying to solve a problem using GA and the chromosome visibility is the fitness function to judge the single level crossover operation (the probability to make mutation is 0%), the result will be as follows: Figure 2-c shows the total system visibility when using GA which equals to 72%.

We got this result because all tasks are arranged in first population by their DL. The task with the lowest DL got the priority to be executed first on CPU. After the first population, the genetic operations will be excuted randomly (crossover and selection) and it will use the fitness function to calculate the CPU visibility for the new chromosome, if it was better than the oldest one then it will be replaced with the new chromosome ignoring the MCT and AT for each task.

Figure 3-a shows the total system CPU utilization average is 80% and this could be a result of the first population sorted by tasks DL with ignoring the MCT and AT for each task. CPUs visibility can be represented in Figure 3-b where CPU No. 1, 2 and 3 execute 4,5 and 4 tasks in sequence.



a) Total system utilization using GA.

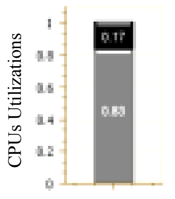


b) CPUs visibility using GA.

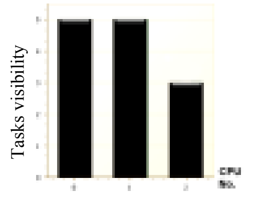
Figure 3. System utilization and CPUs visibility at 80%.

The GA is a random search which means it may have another solution. This solution could be better than the current one.

Figure 4-a represents this behavior when system utilization equals to 83%. Figure 4-b shows the CPUs visibility produced by GA where CPU No: 0, 1 and 2 is 5, 5 and 3 tasks in sequence.



a) Total system utilization using GA.

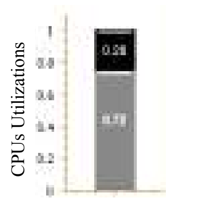


b) CPUs visibility using GA.

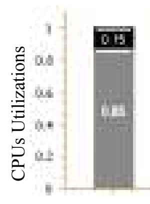
Figure 4. System utilization and CPUs visibility at 83%.

4.3. Genetic Algorithm Crossover Operation

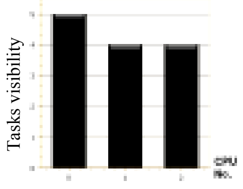
The genetic operations are: Selection, crossover and mutation. The simulator ignores the probability of mutation operation and it focuses on selection and crossover operation and how these operations affect the optimal solution.



a) Total system visibility using multi-point crossover operation through GA.



b) Total system utilization using multi-point crossover operation through GA.



c) CPUs visibility using multi-point crossover operation through GA.

Figure 5. Two types of crossover operations.

There are two types of crossover operations, GA will try to solve the last problem (3 CPU, 18 tasks) by applying multi point crossover operation using GA.

Figure 5-a represents the total system visibility 72% which is the same result as in the single point crossover operation. Figure 5-b shows that the total system utilization is 85%. Figure 5-c shows visibility for each CPU, where CPU No. 0, 1 and 2 can execute the total number of tasks 5, 4 and 4 in sequence .

The closed results that we got from the single point and multi point crossover operations are produced because of the population sorted by tasks' DL .

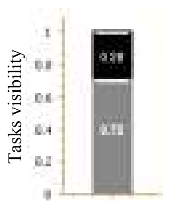
5. Priority Genetic Algorithm

The GA guarantees more visibility of tasks than EDF algorithm as we mentioned in the previous chapter. But, we need to demonstrate the impact of sorting first population on the optimal solution in a different way that the normal GA uses.

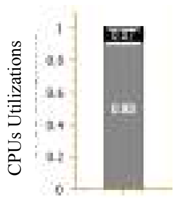
We will find an alternative task DL to sort the first population and see the new effect on the optimal solution where there may be probability to increase the total system visibility and utilization in optimal solution.

5.1. Enhance Genetic Algorithm

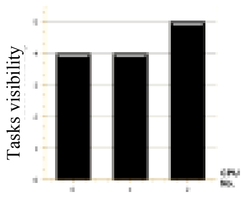
If the property of task MCT is mixed with DL to sort first population, the following equation (P=(DL-MCT)*DL) calculates the new task property called Priority (P). After sorting the first population by priority of tasks we call this algorithm Priority Genetic Algorithm (PGA). The output of the previous problem that mentioned in section 4 shown in the next figures. Figure 6-b shows the system visibility increased beyond the EDF and equal the GA which is 72% while Figure 6-b shows the system utilization that gives 93% and Figure 6-c shows the CPUs visibility that gives CPU No: 0, 1 and 2 values equals to 4, 4 and 5 in sequence. This optimal solution is better than the EDF and GA. The reason behind that will be discussed in the next example.



a) Total System visibility using PG algorithm.



b) Total system utilization using PG algorithm.



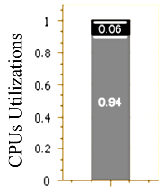
c) CPUs visibility using PG algorithm.

Figure 6. System visibility using PG algorithm.

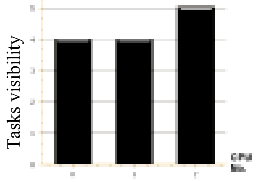
Suppose that two tasks T1 and T2 have DLs 7 and 8, MCTs 1 and 5, ATs 3 and 1 in sequence and the current time in CPU is 0. In this case the scheduler will calculate the priority through the equation P=(DL-MCT)*DL. This gives P values that equals to 42 and 24, respectively. The scheduler then starts with the task T2 because its P is less than T1 so it will start first, the CPU will stay idle until CT equals to 1 and the CT will be equals to CT+MCT for T2. After finishing with T2, the CT will be equals to 6, the CPU will execute T1, where DL for T1>CT, CT>AT and also CT+MCT for T1≤ DL for T1. As a result, the system utilization and visibility will be increased because the CPU will not be idle until CT reaches the AT of T1.

It is concluded from above that the effect of sorting first population on optimal solution could be noticed, but we should remember that the GA operations are consisted of selection, crossover and mutation that are repeated for N time (iteration).

Increasing iteration in GA may be produces a better optimal solution but, this depends on available time that can repeat the GA operations. In other words, GA could reach the optimal solution that PGA reached by increasing the iteration.



a) Total system utilization using PG algorithm.



b) CPUs visibility using PG algorithm

Figure 7. New optimal solution results.

Figures 7-a and 7-b shows the new optimal solution, where Figure 7-a shows the total system utilization increasing and reaches 94% and Figure 7-b shows the CPUs visibility that gives CPU No: 0,1 and 2 values equals to 4, 4 and 5 in sequence.

Before explaining the effect of GA operation on optimal solution, we should discuss the fitness function that takes the responsibilty to accept or refuse the new chromosome. In all previous examples, we assumed that the total fitness function is already specified to accept new chromosomes, then the previous optimal solutions resulted as shown.

Table 1 shows the optimal solutions when we change the fitness function to be single for problem consists of 18 tasks, 3 processors, the RN is supposed to be equals to 20 ns and the total number of iteration for repeating the genetic operation is 100 ns, where the single fitness means every chromosome after crossover operation selected by its fitness value, if this fitness is equal to or greater than the older one, it will be accepted in the population, and if any other value of fitness is less than the oldest chromosome, the new chromosome will be ignored.

Table 1. CPUs visibility using PG algorithm.

	Genetic Algorithm GA							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
Total chromosome fitness	85%	72%	80%	72%	81%	72%	82%	72%
Single chromosome fitness	75%	67%	75%	67%	75%	67%	75%	67%
	Priority Genetic Algorithm PGA							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
Total chromosome fitness	93%	72%	93%	72%	93%	72%	93%	72%
Single chromosome fitness	90%	56%	90%	56%	90%	56%	90%	56%

From the previous table, the average of utilization decreased from 82% to 75% after updating the fitness function and visibility decreased from 72% to 67% using GA, also when using PGA the average of system utilization decreased from 83% to 80% and the average of system visibility from 72% to 56%, however, the fitness function can be used to enhance the GA.

In the next sections, the effects of fitness function will be considered to produce optimal solution and to try the best fitness function that maximize the optimal solution.

In this chapter, there are three methods to accept new chromosomes in population and take them in algorithm consideration, these methods are:

1. *New Total Fitness Value is Best*: In this method, the parent chromosomes fitness value will be compared with the new total fitness value of child chromosomes, if it was greater or equal to the oldest value, it would accept the new chromosomes to be in the population, in this case the oldest one will be deleted.
2. *New Single Fitness Value Best*: The fitness value for the new chromosomes here should be greater than or equal to the oldest value for each chromosome to be accepted in the population, if any fitness value for the new chromosomes is less than the oldest fitness value, the new chromosomes

will never be considered in the algorithm.

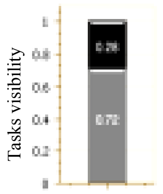
3. *Do Not Care Fitness Value*: The couple of new chromosomes enter the population and delete the oldest one every time after the crossover operation, in this case the fitness value will be ignored.

5.2. Genetic Algorithm Operation

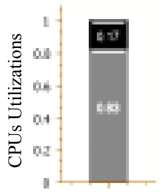
The proposed algorithm will ignore the probability of mutation operation and it focuses on the selection and crossover operations.

5.2.1. Multi Point Crossover Operation vs. Priority Genetic Algorithms

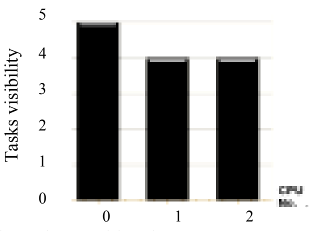
When the scheduler uses the PGA in the same problem (18 tasks, 3 CPUs), it will produce optimal solution with total system visibility equals to 67% as shows in Figure 8-a, total system utilization equals to 94% as in Figure 8-b and the CPU visibility for CPU No: 0, 1, 2 are 4, 3 and 5 in sequence as in Figure 8-c.



a) Total system visibility using multi-point crossover operation through PGA.



b) Total system utilization using multi-point crossover operation through PGA.



c) CPUs visibility using multi-point crossover operation through PGA.

Figure 8. Optimal solution using PGA.

This result shown in Figures 8-a, 8-b and 8-c because the first population is sorted by the task priority that is combination of main task property (MCT and DL) the result came out equals or less than the result that came out from the single point crossover operation, therefore, the proposed algorithm PGA will ignore the multi point crossover operation.

5.2.2. Selection Operation

Selection is one of the genetic operations that is responsible for determining the parent chromosome to enhance it by using crossover operation. In proposed method (PGA), there are three types to select the desired chromosomes:

- *Random Selection*: This technique depends on random number that represents chromosomes number where the minimal one is zero (0) and the maximum one is the number of chromosomes subtract one (1) represented in following equation *0≤ Random≤ number of ckomosomes-1*.
- *Incremental Selection*: This type of selection method selects chromosomes based on the iteration number, where in iteration number zero (0) will select couple of chromosomes zero, one (0, 1) and in the next iteration it will be in (1, 2) chromosomes until GA iteration is complete.
- *Incremental and Random Selection*: This type of selection method depends on merging the previous two types of selection, where the couple of chromosomes can be selected depending on iteration number and random number where in iteration number zero (0) will select chromosome zero (0) and random chromosome selected randomly from population by equation *0≤ Random≤ number of ckomosomes-1*.

The previous problem (18 tasks, 3 CPUs) will be resolved in different selection operations type, where Table 2 represents the optimal solutions for GA and PGA that can be produced.

Table 2. Affects the type of selection operation on optimal solutions.

	Genetic Algorithm GA							
	Try 1		Try 2		Try 3		Average	
	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility
Random	82%	72%	84%	72%	83%	72%	83%	72%
Incremental	75%	67%	75%	67%	75%	67%	75%	67%
Random and Incremental	81%	72%	80%	72%	83%	72%	81%	72%
	Priority Genetic Algorithm PGA							
	Try 1		Try 2		Try 3		Average	
	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility
Random	93%	72%	93%	72%	93%	72%	93%	72%
Incremental	90%	56%	90%	56%	90%	56%	90%	56%
Random and Incremental	92%	72%	94%	72%	92%	72%	93%	72%

The reason behind this result that we got in the incremental selection is the small search space, where the random is better than incremental but, in the next sections it will fail because the space of search will become bigger and it is hard to find the optimal solution randomly in a large search space.

5.3. Fitness Function

It's a function that calculates the weight of the current solution to decide if it's better than the previous solution. There are no rules for determining how the fitness function calculates this weight (value) but, it should focus on ability to find optimal solution, if the proposed method aims at increasing the system visibility, the fitness function should calculate the visibility for each chromosome within the population before and after crossover operation is being done.

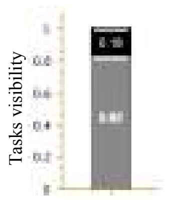
The fitness function that is used in this chapter so far to increas the visibility of the system. In this section the PGA uses two new types of fitness function as follows:

- *Chromosome Visibility*: The value of the fitness function here depends on the total tasks that can finish the execution MCT on resource before reaching it is DL and we used this method to solve all previous examples.

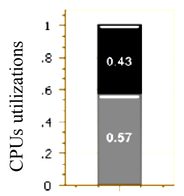
- *Chromosome Utilization*: The value of the fitness function here is depending on the CPU utilization which represented in this schedule.
- *Chromosomes Utilization and Visibility*: The value of the fitness function in this part is a combination of the previous two types as it depends on chromosome visibility and utilization.

If we try solve a real time multi processor tasks scheduling problem consists of 4 CPUs and 40 tasks with RN equals to 200 ns and iteration number is 400 using single point crossover and three types of fitness function by EDF, GA and PGA the optimal solutions can be represented as follows.

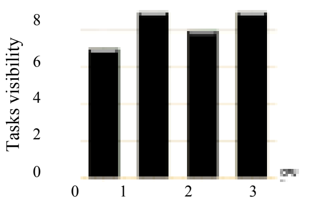
Figure 9-a shows that the total system visibility can be reached using EDF algorithm where system visibility is 82%, total system utilization equals 57% represented in Figure 9-b and CPU No: 0, 1, 2 and 3 can succeed to execute total tasks 7, 9, 8 and 9 in sequence as represented in Figure 9-c.



a) Total system visibility using EDF algorithm



b) Total system utilization using EDF algorithm.



c) CPUs visibility using EDF algorithm.

Figure 9. System results using EDF at 82%.

Table 3. The effect of fitness function on optimal solution using GA and PGA.

	Genetic algorithm							
	Try 1		Try 2		Try 3		Average	
	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility
CPU utilization	60%	82%	58%	82%	57%	82%	58%	82%
CPU visibility	60%	90%	62%	90%	63%	88%	62%	89%
CPU utilization and visibility	62%	88%	63%	88%	63%	88%	63%	88%
	Priority genetic algorithm							
	Try 1		Try 2		Try 3		Average	
	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility	Utilization	Visibility
CPU utilization	63%	82%	66%	82%	65%	78%	65%	81%
CPU visibility	74%	92%	70%	92%	71%	92%	72%	92%
CPU utilization and visibility	72%	92%	72%	92%	70%	90%	71%	91%

When trying to solve the previous problem (40 tasks, 4 CPUs) using GA and PGA, the optimal solutions for system visibility and total system utilization generally increase. Table 3 above shows optimal solutions that it can reach with different types of fitness functions.

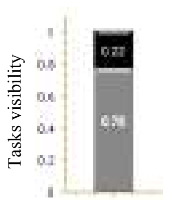
It is obvious from Table 3 that the proposed method PGA has better optimal solution more than GA in different types of fitness function, and it's noticed that the effects of fitness function in optimal solution that the algorithm can reach.

5.4. Scheduler Runing Time SRN

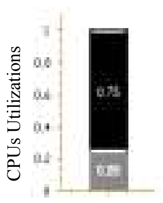
The RT should be known to the scheduler but until now RN effects have no result measures. This section clarifies what happens to scheduling algorithm if the RN is increased.

The EDF, GA and PGA will re-execute the first problem (3 CPUs, 18 tasks and iteration number equals to 240) but, suppose that the 18 tasks arrives before time equals 200 ns (RN equal to 200), where MCT for each task selected randomly is represented by the following equation $0< \text{MCT} \leq 30\text{ns}$.

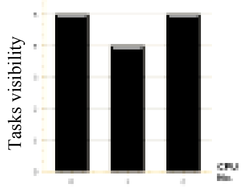
The EDF algorithm can execute 78% from total tasks represented in Figure 10-a. Figure 10-b represent the total system utilization which is equals to 25% and Figure 10-c show the tasks that can be executed successfully on each processor where CPU No: 0,1 and 2 execute the total tasks 5, 4 and 5.



a) Total system visibility using EDF algorithm.



b) Total system utilization using EDF algorithm.

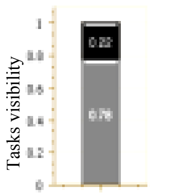


c) CPUs visibility using EDF algorithm.

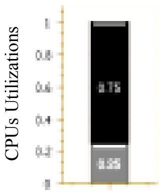
Figure 10. System results using EDF at 78%.

From the previous result, we can say that the difference between the total system visibility and total system utilization is quite large, and that's because EDF algorithm sorts the tasks by its DLs where almost CPUs time is wasted by waiting the system time reaching the AT for other tasks.

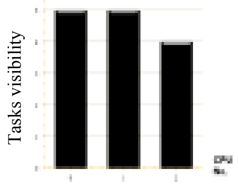
The GA beats the EDF algorithm which gives better optimal solution where the total of system visibility increased to reach 78% that represented in Figure 11-a, as well as the total system utilization increased to reach 25% shown in Figure 10-b where CPU No: 0, 1 and 2 execute the total tasks 5, 4 and 5 in sequence represented in Figure 10-c.



a) Total system visibility using GA.



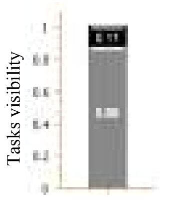
b) Total system utilization using GA.



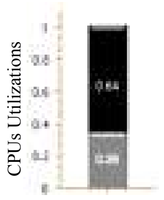
c) CPUs visibility using GA.

Figure 11. System results using GA.

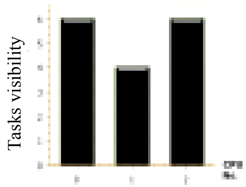
PGA defeats both EDF and GA, as it guarantees more better optimal solution and increase total system visibility to reach 89% as shown in Figure 12-a, and total system utilization increased to reach 36% as shown in Figure 12-b where CPU No: 0, 1 and 2 execute the total tasks 6,4 and 6 in sequence represented in Figure 12-c.



a) Total system visibility using PGA.



b) Total system utilization using PGA.



c) CPUs visibility using PGA.

Figure 12. System results using GA at 89%.

5.5. Ability of Priority Genetic Algorithm to Solve Large Scheduling Problem

The following Table 4 shows the ability of simulator to solve large scheduling problem consists of 2000 tasks, 100 processors, RN equals to 400 ns and the total number of iteration to repeat the genetic operation is 8000.

Table 4. Result of simulater for large scheduling problem.

	Chromosome visibility as fitness function replacement method is total fitness of chromosome is best	
	Utilization	**Visibility**
EDF	55%	86%
GA	62%	92%
PGA	69%	97%

From the Table 4 PGA defeats both EDF and GA, as it guarantees more better optimal solution and increase total system visibility to reach 97%, and total system utilization increased to reach 69%.

5.6. Earliest Deadline First vs. Genetic vs. Priority Genetic Algorithms

In this section we proposed three different real time multi processor tasks scheduling problems, we will try to solve these problems using the EDF, GA and PGAs to improve the visibility and utilization for systems. The proposed problems consist of the following:

- *Problem 1*: This problem consists of 18 tasks, 3 processors, the RN is supposed to be equals to 20 ns and the total number of iteration for repeating the genetic operation is 100 ns.
- *Problem 2*: This problem consists of 40 tasks, 4 processors, the RN is supposed to be equals to 200 ns and the total number of iteration for repeating the genetic operation is 200.
- *Problem 3*: This problem consists of 150 tasks, 8 processors, the RN is supposed to be equals to 250 ns and the total number of iteration for repeating the genetic operation is 200.

The GA and PGA will shows different optimal solutions when use different fitness functions and different replacement methods, but assumes that genetic and PG algorithm use the incremental and random technique as a selection method, the following tables explain all results that can be produced. All tasks properties supposed to prove the following equations:

1. $AT > 0$ for any task in population.
2. $AT < DL$.
3. $AT + MCT \leq DL$.
4. $0 < MCT \leq 30$ *ns*.

5.6.1. Problem 1: 18 Tasks/ 3 Processors

Table 5. Result for problem 1 using chromosome visibility as fitness function and replacement method is total fitness of chromosome is best.

	Problem 1: Chromosome visibility as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	60%	82%	58%	82%	57%	82%	58%	82%
GA	60%	90%	62%	90%	63%	88%	62%	89%
PGA	62%	88%	63%	88%	63%	88%	63%	88%

Table 6. Result for problem 1 using chromosome utilization as fitness function and replacement method is total fitness of chromosome is best.

	Problem 1: Chromosome utilization as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	80%	44%	80%	44%	80%	44%	80%	44%
GA	87%	56%	88%	56%	90%	56%	88%	56%
PGA	90%	56%	87%	56%	87%	56%	87%	56%

Table 7. Result for problem 1 using chromosome visibility and utilization as fitness function and replacement method is total fitness of chromosome is best.

	Problem 1: Chromosome visibility and utilization as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	80%	44%	80%	44%	80%	44%	80%	44%
GA	87%	56%	90%	56%	88%	56%	88%	56%
PGA	88%	56%	87%	56%	90%	56%	88%	56%

Table 8. Result for problem 1 using chromosome visibility as fitness function and replacement method is single fitness of chromosome is best.

	Problem 1: Chromosome visibility as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	80%	44%	80%	44%	80%	44%	80%	44%
GA	80%	44%	80%	44%	80%	44%	80%	44%
PGA	86%	50%	86%	50%	86%	50%	86%	50%

Table 9. Result for problem 1 using chromosome utilization as fitness function and replacement method is single fitness of chromosome is best.

	Problem 1: Chromosome utilization as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	80%	44%	80%	44%	80%	44%	80%	44%
GA	87%	50%	86%	50%	88%	50%	87%	50%
PGA	86%	50%	86%	50%	86%	50%	86%	50%

Table 10. Result for problem 1 using chromosome visibility and utilization as fitness function and replacement method is single fitness of chromosome is best.

	Problem 1: Chromosome visibility and utilization as fitness replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	80%	44%	80%	44%	80%	44%	80%	44%
GA	80%	44%	80%	44%	80%	44%	80%	44%
PGA	86%	50%	86%	50%	86%	50%	86%	50%

From the previous six tables, the average of the total system utilization and visibility that we got from GA and PGA are almost the same.

5.6.2. Problem 2: 40 Tasks/ 4 Processors

Table 11. Result for problem 2 using chromosome visibility as fitness function and replacement method is total fitness of chromosome is best.

	Problem 2: Chromosome visibility as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	80%	44%	80%	44%	80%	44%	80%	44%
GA	62%	88%	63%	88%	62%	88%	62%	88%
PGA	69%	92%	68%	92%	68%	92%	68%	92%

Table 12. Result for problem 2 using chromosome utilization as fitness function and replacement method is total fitness of chromosome is best.

	Problem 2: Chromosome utilization as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	57%	82%	57%	82%	57%	82%	57%	82%
GA	53%	82%	61%	88%	57%	82%	57%	84%
PGA	63%	82%	67%	85%	67%	88%	66%	85%

Table 13. Result for problem 2 using chromosome visibility and utilization as fitness function and replacement method is total fitness of chromosome is best.

	Problem 2: Chromosome visibility and utilization as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	57%	82%	57%	82%	57%	82%	57%	82%
GA	64%	90%	63%	88%	61%	88%	57%	84%
PGA	68%	92%	71%	92%	71%	92%	70%	92%

Table 14. Result for problem 2 using chromosome visibility as fitness function and replacement method is single fitness of chromosome is best.

	Problem 2: Chromosome visibility as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	57%	82%	57%	82%	57%	82%	57%	82%
GA	57%	82%	57%	82%	57%	82%	57%	82%
PGA	67%	88%	67%	88%	67%	88%	67%	88%

Table 15. Result for problem 2 using chromosome utilization as fitness function and replacement method is single fitness of chromosome is best.

	Problem 2: Chromosome utilization as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	57%	82%	57%	82%	57%	82%	57%	82%
GA	59%	82%	63%	88%	60%	82%	61%	84%
PGA	69%	88%	70%	88%	69%	90%	69%	89%

Table 16. Result for problem 2 using chromosome visibility and utilization as fitness function and replacement method is single fitness of chromosome is best.

	Problem 2: Chromosome visibility and utilization as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	57%	82%	57%	82%	57%	82%	57%	82%
GA	59%	82%	59%	82%	59%	82%	59%	82%
PGA	67%	88%	67%	88%	67%	88%	67%	88%

The PGA beats the GA in the previous six tables. We got the best result from PGA when we've used chromosome visibility and utilization as fitness function and replacement method as total fitness of chromosome is best.

5.6.3. Problem3: 150 Tasks/ 8 Processors

Table 17. Result for problem 3 using chromosome visibility as fitness function and replacement method is total fitness of chromosome is best.

	Problem 3: Chromosome visibility as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	59%	69%	59%	69%	59%	69%	59%	69%
GA	65%	71%	66%	73%	65%	73%	65%	72%
PGA	76%	76%	75%	75%	77%	77%	76%	76%

Table 18. Result for problem 3 using chromosome utilization as fitness function and replacement method is total fitness of chromosome is best.

	Problem 3: Chromosome utilization as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	59%	69%	59%	69%	59%	69%	59%	69%
GA	61%	68%	62%	69%	57%	67%	60%	68%
PGA	69%	69%	73%	70%	73%	71%	71%	70%

Table 19. Result for problem 3 using chromosome visibility and utilization as fitness function and replacement method is total fitness of chromosome is best.

	Problem 3: Chromosome visibility and utilization as fitness function replacement method is total fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	59%	69%	59%	69%	59%	69%	59%	69%
GA	66%	73%	67%	73%	68%	74%	67%	73%
PGA	75%	74%	76%	75%	73%	74%	75%	74%

Table 20. Result for problem 3 using chromosome visibility as fitness function and replacement method is single fitness of chromosome is best.

	Problem 3: Chromosome visibility as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	59%	69%	59%	69%	59%	69%	59%	69%
GA	56%	66%	56%	66%	56%	66%	56%	66%
PGA	71%	72%	69%	71%	71%	72%	70%	71%

Table 21. Result for problem 3 using chromosome utilization as fitness function and replacement method is single fitness of chromosome is best.

	Problem 3: Chromosome utilization as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	59%	69%	59%	69%	59%	69%	59%	69%
GA	62%	70%	63%	71%	61%	71%	62%	71%
PGA	82%	68%	84%	70%	84%	70%	83%	70%

Table 22. Result for problem 3 using chromosome visibility and utilization as fitness function and replacement method is single fitness of chromosome is best.

	Problem 3: Chromosome visibility and utilization as fitness function replacement method is single fitness of chromosome is best							
	Try 1		**Try 2**		**Try 3**		**Average**	
	Utilization	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**	**Utilization**	**Visibility**
EDF	59%	69%	59%	69%	59%	69%	59%	69%
GA	56%	66%	56%	66%	56%	66%	56%	66%
PGA	71%	72%	69%	71%	71%	72%	70%	71%

The PGA beats the GA in the previous six tables. We got the best result from PGA when we've used chromosome visibility as fitness function and replacement method as total fitness of chromosome is best.

From the previous tables, The GA defeats the EDF algorithm for all supposed systems and the PGA defeat both EDF and GA in these systems. These results occur because the proposed algorithm focused on the first population and how it should be sorted and does not stop to sort it by tasks DL where MCT involved into calculating additional property represented as P where the first population is sorted by it.

After all, We got the best result from PGA and GA when we have used chromosome visibility or visibility and utilization as fitness function and replacement method as total fitness of chromosome is best. As this organize the search space.

6. Conclusion and Recommendations

At the end, we can say that the EDF, GA and PG algorithms are techniques created to solve the real time multi processor tasks scheduling problems as genetic defeats the EDF algorithm, and the PG defeats EDF and GAs.

The PGA is a random search technique that applies some operations to repeat N iterations. These operations can be evaluated by fitness function to determine the weight of the new child is better than its parent. It is worth mentioning that most of algorithms ignore the effects of sorting the first population on the optimal solution.

Results prove that, the fitness function has a major effect on enhancing the optimal solutions as needed. It can be used to improve the system utilization, system visibility or both.

Table 23 shows the overall average results for the discussed three problems. From this table we see that suggested algorithm PGA has better performance for three discussed cases.

Table 23. Overall average of optimal solutions for the discussed three problems.

	Problem 1		**Problem 2**		**Problem 3**	
	Overall utilization	**Overall visibility**	**Overall utilization**	**Overall visibility**	**Overall utilization**	**Overall Visibility**
EDF	58%	82%	80%	44%	59%	69%
GA	80.83%	56.5%	58.83%	84%	61%	69.33%
PGA	82.66%	58.3%	67.83%	89%	74.16%	72%

Finally, we can conclude that PGA and GA give better optimal solution using the following parameters with the respected values:

- Fitness function with chromosome visibility or visibility and utilization.
- Selection method with incremental and random.
- Replacement method with total fitness of chromosome is best.

The performance of the proposed algorithm (PGA) may be improved by:

1. Organizing and increasing the search space, through using a combination of incremental and random selection operation.
2. When the SRN is large in system respectively to the total task number, if may be better to use EDF algorithm because it's going to give an approximately the same result of optimal solution that is given by GA and PGA, and the EDF has a less complexity.
3. Combining the PGA with Partially Matched Crossover (PMX) may give better results.

References

[1] Abraham A., Dahal K., Hossain A., Daradoumis A., Varghese B., and Xhafa F., "Scheduling in Multi processor System Using Genetic Algorithms," *in Proceedings of the 7th Computer Information Systems and Industrial Management Applications*, Ostrava, Czech Republic, pp. 281-286, 2008.

[2] Agarwal G., Gupta S., and Kumar V., "Task Scheduling in Multi Processor System Using Genetic Algorithm," *in Proceedings of the 2nd International Conference on Machine Learning and Computing*, Bangalore, Karnataka, pp. 267-271, 2010.

[3] Ceyda O. and Ercan M., "A Genetic Algorithm for Multilayer Multi Processor Task Scheduling," *in Proceedings of IEEE Region 10 Conference*, vol. 2, pp. 168-170, 2004.

[4] Cheng S. and Huang Y., "Dynamic Real-Time Scheduling for Multi Processor Tasks Using Genetic Algorithm," *in Proceedings of the 28th Annual International Computer Software and Applications Conference*, Hong Kong, China, vol. 1, pp. 154-160, 2004.

[5] Dandass Y., "Genetic List Scheduling for Soft Real-Time Parallel Applications," *in Proceedings of Congress on Evolutionary Computation*, Portland, USA, vol. 1, pp. 1164-1171, 2004.

[6] Daoud M. and Kharma N., "An Efficient Genetic Algorithm for Task Scheduling in Heterogeneous Distributed Computing Systems," *in Proceedings of IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp. 3258-3265, 2006.

[7] Deshpande N. and Kamalapur S., "Efficient CPU Scheduling: A Genetic Algorithm based Approach," *in Proceedings of International Symposium on Ad-Hoc and Ubiquitous Computing*, Surathkal, India, pp. 206-207, 2006.

[8] Galvin P., Gagne G., and Silberschatz A., *Operating System Concepts*, Wiley Knowledge for Generations, USA, 2005.

[9] Graham R., "Static Multi-processor Scheduling with Ant Colony Optimisation and Local Search," *Master of Science*, University of Edinburgh, UK, 2003.

[10] Hnang Z. and Wu Z., "A Deadlock-Free Scheduling Method for Automated Manufacturing Systems Using Dynamic-Edge Graph with Tokens," *in Proceedings of the IEEE International Conference on Control Applications*, Taipei, Taiwan, vol. 2, pp. 1398-1403, 2004.

[11] Ilavarasan E. and Thambidurai P., "Genetic Algorithm for Task Scheduling on Distributed Heterogeneous Computing System," *World Academy of Science, Engineering and Technology*, vol. 1, no. 3, pp. 233-242, 2006.

[12] Jie l., Ruifeng G., and Zhixiang S., "The Research of Scheduling Algorithms in Real-time System," *in Proceedings of International Conference on Computer and Communication Technologies in Agriculture Engineering*, Chengdu, China, vol. 1, pp. 333-336, 2010.

[13] Kazem A., Seifzadeh H., Kargahi M., Movaghar A., and Lotfi S., "Maximizing the Accrued Utility of an Isochronal Soft Real-Time System Using Genetic Algorithms," *in Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science*, Shanghai, China, pp. 65-69, 2009.

[14] Kumar R. and Gill S., "An Impact of Crossover Operator on the Performance of Genetic Algorithm Under Operating System Process Scheduling Problem," *in Proceedings of International Conference on Communication Systems and Network Technologies*, Katra, Jammu, pp. 704-708, 2011.

[15] Lin C. and Luh J., "Scheduling of Parallel Computation for a Computer-Controlled Mechanical Manipulator Systems," *Man and Cybernetics*, vol. 12, no. 2, pp. 214-234, 1982.

[16] Mahmood A., "A Hybird Genetic Algorithm for Task Scheduling in Multi Processor Real-Time Systems," *in Proceedings of Studies in Informatics and Control*, pp. 207-218, 2009.

[17] Man L. and Sai M., "A Genetic Algorithm for Energy Aware Task Scheduling in Heterogeneous Systems," *Parallel Processing Letters*, vol. 15, no. 4, pp. 439-449, 2005.

[18] ManChon U., Chiahsun H., Funk S., and Rasheed K., "A Genetic Algorithm Based Real-Time System Scheduler," *in Proceedings of IEEE Congress on Evolutionary Computation*, New Orleans, USA, pp. 886-893, 2011.

[19] Montana D., Bidwell G., and Moore S., "Using Genetic Algorithms for Complex, Real-Time Scheduling Applications," *in Proceedings of IEEE Network Operations and Management Symposium*, New Orleans, USA, vol. 1, pp. 245-248, 1998.

[20] Pooranian Z., Harounabadi A., Shojafar M., and Hedayat N., "New Hybrid Algorithm for Task Scheduling in Grid Computing to Decrease Missed Task," *World Academy of Science, Engineering and Technology*, vol. 5, no. 7, pp. 959-963, 2011.

[21] Rashtbar S., Isazadeh A., and Khanly L., "A New Hybrid Approach for Multi processor System Scheduling with Genetic Algorithm and Tabu Search (HGTS)," *in Proceedings of the* 3rd *International Conference on Information Sciences and Interaction Sciences*, Chengdu, China, pp. 626-631, 2010.

[22] Sachi G., Gaurav A., and Vikas K., "Task Scheduling in Multi Processor System Using Genetic Algorithm," *in Proceedings of the 2nd International Conference on Machine Learning and Computing*, Bangalore, Karnataka, pp. 267-271, 2010.

[23] Suranauwarat S., "A CPU Scheduling Algorithm Simulator, Frontiers In Education Conference - Global Engineering," *Knowledge Without Borders*, vol. 7, no. 37, pp. 19-24, 2007.

[24] Zhong Y. and Yang J., "A Genetic Algorithm for Tasks Scheduling in Parallel Multi Processor Systems," *in Proceedings of the International Conference on m Machine Learning and Cybernetics*, vol. 3, pp. 1785-1790, 2003.

[25] Zhu C., Dai S., and Zhi L., "Task Matching and Scheduling by Using Self-Adjusted Genetic Algorithms," *in Proceedings of the 5th IEEE International Conference on Cognitive Informatics*, Beijing, China, vol. 3, pp. 908-911, 2006.

Mohammad Ababneh received his PhD in computer engineering from Cairo University, Egypt in 2000. He is an associate professor of computer engineering, he is an instructor in Computer Information Systems, Balqa Applied University, Jordan. Teaching different subjects like, microprocessors, computer organization, computer architecture, image processing, algorithms, compilers, programming, etc., he also held administration positions as dean, vice dean, dean assistance, and head of department. Published around 17 papers in different computer deciplines.

Salama Hassan received his bachelor degree in computer science from prince Abdullah Bin Ghazi Faculty of Information Technology-Amman, Jordan in 2008. He has also received his from the same school in 2012. Currently, he is working for the Health Insurance Directorate, Jordan as ORACLE programmer.

Sulieman Bani-Ahmad received his BSc degree in electrical engineering/ computer engineering from the Department of Electrical Engineering, Jordan University of Science and technology in 1999. He received an MSc in computer science from the School of Information Technology at Al-albayt University in Jordan, in 2001. He received his PhD degree in computing and information systems from the Department of Electrical Engineering and Computer Science at Case Western Reserve University, Cleveland-Ohio, USA, in 2008. He is presently a professor at Al-Balqa Applied University, Jordan. His research interests include web-computing and online literature digital libraries. More specifically, he is interested in social network analysis of literature citation graphs, domain-specific citation-behavior in literature citation networks, and research development models of literature. He has also, works in the area of e-learning and technology-based teaching. Finally, he works in the area of parallel computing. More specifically, he worked on the topics of processor allocation and job scheduling.