# Fittest Job First Dynamic Round Robin (FJFDRR) scheduling algorithm using dual queue and arrival time factor: a comparison



# Fittest Job First Dynamic Round Robin (FJFDRR) scheduling algorithm using dual queue and arrival time factor: a comparison

**Jezreel Ian C. Manuel[1], Rey Benjamin M. Baquirin[2], Kier Sostenes Guevara[3], and Dionisio R. Tandingan Jr[4]**
University of the Cordilleras, Baguio City, Philippines

[1]jezreelianmanuel@gmail.com
[2]reybenbaq@gmail.com
[3]kier.guevara @gmail.com
[4]dion_tand@yahoo.com

**Abstract**. The Fittest Job First Dynamic Round Robin (FJFDRR) was introduced as a CPU scheduling algorithm whose performance evaluation reduces the number of context switches (CS), average waiting time (AWT), and average turnaround time (ATAT) of processes in a single CPU environment. In this paper, we explored improvements on the FJFDRR by including the process arrival time as an algorithmic factor implemented using a dual queue. We then compared the performance of the proposed algorithm called enhanced Fittest Job First Dynamic Round Robin (eFJFDRR) as with the FJFDRR algorithm together with the other CPU scheduling algorithms. Trial results showed that eFJFDRR scheduling algorithm performed better in reducing average waiting time, average turnaround time, and average response time in some cases. It was also found to balance the number of context switches of the processor during execution.

## 1. Introduction

CPU scheduling determines the order in which processes are executed in the CPU. A computer system that implements an effective scheduling algorithm is one that optimizes CPU utilization, minimizes response time, and maximizes throughput [1]. As such, scheduling algorithms have remained to be the topic of interest in computer architecture and operating systems researches. With modern operating systems becoming more complex, there is a need to continually improve CPU allocation through scheduling algorithms to maximize system throughput, and CPU utilization [2].

This paper focuses on the improvement of the existing scheduling algorithms based on the Best Job First (which uses a fit factor 'f' where 'f' is given by the summation of priority, arrival time and burst time of a process) and the Fittest Job First Dynamic Round Robin (which combines the best job first (BJF) algorithm and other scheduling algorithms starting with the lowest Fit Factor 'f' where 'f' is given by the constant values of user priority weight and burst time priority weight.). The improvement of the two algorithms will be called as enhanced Fittest Job First Dynamic Round Robin (eFJFDRR). This algorithm implements using dual ready queues and arrival time factor to satisfy the scheduling criteria which decreases the average turnaround time, average waiting time, average response time and context switching. The proposed eFJFDRR scheduling

algorithm is compared to the other CPU scheduling namely first come first serve (FCFS), shortest job first (SJF), round robin (RR), and best job first (BJF) algorithms.

## 2. Review of Related Literature

Several CPU scheduling algorithms have different properties and may favor one class of processes over another. Users must consider the characteristics of these algorithms to ensure a better performance metrics. Many criteria have been suggested for comparing CPU scheduling algorithms. The following criteria include:

***CPU utilization:*** A user want to keep the CPU as busy as possible. It may range from 0 to 100%. In real time system, it suits range from 40% (for a lightly loaded system) to 90% (for heavily loaded system).

***Throughput:*** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes completed per time unit or throughput. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.

***Turnaround Time:*** From the submission time of a process to its completion time is turnaround time. It refers to the total time period spends waiting to get into memory, waiting in the ready queue, executing in the CPU and doing input/output operations.

***Waiting Time:*** The CPU scheduling algorithm does not affect the amount of the time during which process execute or does input/output. It affects only the amount of time that a process spends waiting in the ready queue. Waiting time refers to the sum of time periods spends waiting in the ready queue.

***Response Time:*** It refers to the time from the submission of a request until the first response is produced called response time. It is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device.

***Context Switch:*** The process of storing the state of a process or of a thread, so that it can be restored and execution resumed from the same point later. This allows multiple processes to share a single CPU, and is an essential feature of a multitasking operating system [3].

Most of the newly proposed scheduling algorithms today are combinations or improvements of basic pre-emptive and non-pre-emptive solutions like First Come First Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR). This is primarily done to overcome the limits posed by each individual algorithm. For example, FCFS scheduling solely relies on the execution of processes in the order they arrive in the ready queue, hence increasing the waiting time of all processes with short burst-times that follow processes with longer burst-times [4]. Conversely, SJF scheduling executes the shortest processes first but leaves longer processes starved [5]. Priority Scheduling executes processes with the highest priority first and starves processes with lower priority [6]. RR, on the other hand, executes each process equally on a constant time quantum, which may reduce system throughput if the time quantum is too small [7].

The limitations mentioned above motivate many programmers to come up with hybrid algorithms for process scheduling. Two such algorithms are imperative to this study. Foremost is the Best Job First (BJF) CPU Scheduling Algorithm [8] which was used as a benchmark for comparison against the proposed algorithm as shown in Section 3 of this paper. BJF uses a computed Factor 'f' to schedule processes where 'f' is given by:

$$f = Priority + Arrival\ Time + Burst\ Time \quad \text{----- (equation1)}$$

BJF schedules processes starting from the lowest 'f' score, then executes them in order, non-pre-emptively. Second is the Fittest Job First Dynamic Round Robin [8] (FJFDRR), an algorithm that combines BJF with other algorithms [4, 8] and schedules processes in order starting with the lowest Fit Factor 'f' score as given by:

$$f = UP * UW + SP * BW \quad \text{----- (equation 2)}$$

Where UP = User Priority, UW = User Priority weight, SP = Shorter Burst time Priority, and BW = Burst time Priority Weight. As user Priority weight(UW) was deemed to be more important than burst time priority weight(BW) [8], UW and BW have constant values of 60% and 40% respectively assuming that all processes have same arrival time. Once ordered, the algorithm

then executes the processes on Dynamic Round Robin; computing variable time quantum based on the median of all processes' remaining burst time [9]. A performance evaluation done on the FJFDRR algorithm in 2011 yielded a decrease in the number of context switches, average waiting time, and average turnaround time [10] when compared against the Priority Based Static Round Robin (PBSRR) algorithm. However, the experimental test cases used did not factor in the arrival time for each process and the researchers recommended further studies be done on the algorithm to incorporate it.

Taking on the recommendation, the researchers present an improvement on the FJFDRR scheduling algorithm by incorporating the arrival time of processes into the algorithm and managing them through multiple queues. The researchers refer to our algorithm as eFJFDRR henceforth.

## 3. Methods

The methodology for eFJFDRR comprises the following:

(1) The original Fit Factor formula 'f' as given by

**f = (UP * 0.6) + (SP * 0.4)** ----- (equation 3)

(2) The inclusion of each process' Arrival Time as an algorithmic factor;

(3) The use of an additional queue to handle Newly Arrived processes; and

(4) A dynamic time quantum [10] implemented for 1 RR cycle per queue.

The algorithm can be described as follows:

(1) Before execution starts on the processes in the ready queue, each process' Fit Factor 'f' is computed using equation 3.

(2) They are rearranged ascendingly in the queue and the time quantum is computed by getting the median of all processes' remaining burst times.

(3) The queue starts process execution via RR using the computed time quantum.

(4) Meanwhile, all processes that enter the CPU while the first queue is executing are added to a separate queue.

(5) When the first queue completes 1 cycle of RR, it then switches to the other queue, and treats it as the new ready queue starting again with step 1 above with the difference that when it gets to step 4, the algorithm does not create another queue to add newly arrived processes, instead it uses the first queue (which is not executing at this point) and inserts the newly arrived processes there. The algorithm is repeated until no processes remain to be executed in either queue. Evidently, eFJFDRR relies heavily on the alternating execution of two queues via dynamic RR. Further illustration on how the algorithm works is shown below:

### *3.1. Pseudocode of the eFJFDRR algorithm*

Let Q1 and Q2 be the queues to be executed alternately. Let TQ be the time quantum. Let NA be the newly arrived processes.

1. Let Q1 be the ready queue and Q2 the waiting queue.
2. Add the first process to arrive at the ready queue.
3. If there are processes with the same arrival time as the first process, add it to the ready queue.
4. Calculate the fit factor of each process in the ready queue using Equation (3).
5. Sort the ready queue in ascending order by the fit factor.
6. Set TQ = median of the remaining burst time of the ready queue.
7. Execute the ready queue for 1 cycle of RR.
8. While (ready queue is executing) {
   add NA to the waiting queue.
   remove process with no remaining burst time from queue.
   }
9. If (ready queue and waiting queue is empty)
   repeat steps 1-9.

```
Else if (waiting queue is empty)
        repeat steps 4-9.
Else if (waiting queue is not empty)
{
    If (Q1 = ready queue)
        {Let Q2 be the ready queue and Q1 be the waiting queue}
    Else
      {Let Q1 be the ready queue and Q2 be the waiting queue}
      repeat step 4-9
}
```

*3.2. Illustration of the eFJFDRR algorithm*

To further illustrate how the algorithm works, we present a simulation. Table 1 below shows 6 processes with arrival time, priority, and burst time all known before CPU execution.

**TABLE 1**. SAMPLE PROCESSES

| Process | Arrival Time | Priority | Burst Time | Fit Factor |
|---|---|---|---|---|
| P0 | 1 | 1 | 20 | 8.6 |
| P1 | 10 | 2 | 10 | 5.2 |
| P2 | 15 | 1 | 15 | 6.6 |
| P3 | 17 | 3 | 5 | 3.8 |
| P4 | 22 | 1 | 5 | 2.6 |
| P5 | 25 | 2 | 8 | 4.4 |

Process P0 arrived first with the arrival time of 1. Since no other processes have the same arrival time as P0, only P0 will be added to Q1. Since Q1 has only 1 process, TQ is 20. Q1 will be executed for 1 round. While Q1 is executing, processes P1, P2, and P3 arrive. They will be added to Q2. When Q1 is finished executing, P0 has no remaining burst time so it will be removed from the queue. The fit factor for P1, P2 and P3 are 5.2, 6.6, and 3.8 respectively. The TQ for Q2 is 10 which is the median of the remaining burst time of Q2. The process with the least fit factor will be scheduled first hence the order will be P3, P1, and P2. Q2 will be executed for 1 round. While Q2 is executing, processes P4 and P5 arrive. They will be added to Q1 since Q2 is executing. When Q2 is finished executing, only P2 remains with a remaining burst time of 5. The fit factor for P4 and P5 are 2.6 and 4.4 respectively. The time quantum for Q1 is 6. Q1 will be executed. Since P4 has a lower fit factor, it will be scheduled first. While Q1 is executing, there are no processes to add to Q2 since there are no processes that arrive. When Q1 finished, only P5 remains with a remaining burst time of 2. Q2 will begin executing with TQ as 5. When Q2 finished executing, it will be empty and only Q1 remains with 1 process which is P5. Q1 will begin executing with TQ as 2. When Q1 finished executing, there will be no remaining process to execute.

## 4. Test Cases

All experimental test cases were performed with the following assumptions: (1) Processes are executed in a single processor. (2) All the processes are CPU bound. (3) All parameters, namely total number of processes, arrival time, burst time, and priority for each process, are known before CPU execution starts. (4) BJF, FCFS, SJF, and RR are used as benchmark algorithms. RR will have a time quantum of 15 as per the performance evaluation of the original FJFDRR [9].

The metrics used for algorithm comparison are the average turnaround time (ATAT), average waiting time (AWT), average response time (ART) and number of context switches (CS). ATAT is the mean time from submission to completion of processes. AWT is the average amount of

time processes are in a ready state waiting to be picked up for execution. ART is the average time between the submission of the processes and the first response to that request. CS is the number of times the CPU changes from one process to another. All of these metrics are evaluated per algorithm with the lower scores signifying better performance.

***Test Case 1:*** We assumed 5 processes. Processes with longer burst times arrive earlier and have lower priority while processes with shorter burst times arrive later and have higher priority. Table 2 presents the test case parameters including the computed Fit Factor for each process.

**TABLE 2**. TEST CASE 1

| Process | Arrival Time | Priority | Burst Time | Fit Factor |
|---|---|---|---|---|
| P0 | 1 | 10 | 100 | 46 |
| P1 | 20 | 5 | 40 | 19 |
| P2 | 31 | 4 | 17 | 9.2 |
| P3 | 54 | 3 | 12 | 6.6 |
| P4 | 70 | 1 | 6 | 3 |

Table 3 shows the comparative results of eFJFDRR against the algorithms assumed as benchmarks.

**TABLE 3.** E FJFDRR RESULT ON TEST CASE 1

| Algorithm | TQ | ATAT | AWT | ART | CS |
|---|---|---|---|---|---|
| eFJFDRR | 100, 14, 14, 12 | 95.4 | 60.4 | 57 | 7 |
| BJF | N/A | 95.8 | 60.8 | 60.8 | 5 |
| RR | 15 | 91 | 91.2 | 19.6 | 14 |
| SJF | N/A | 92.6 | 57.6 | 57.6 | 5 |
| FCFS | N/A | 114 | 79 | 79 | 5 |

eFJFDRR performed better than the remaining algorithms, except SJF, in ATAT and AWT in this test case as shown in Figure 1 below. It also scored better in ART, coming in second to RR. CS score was average in this case.

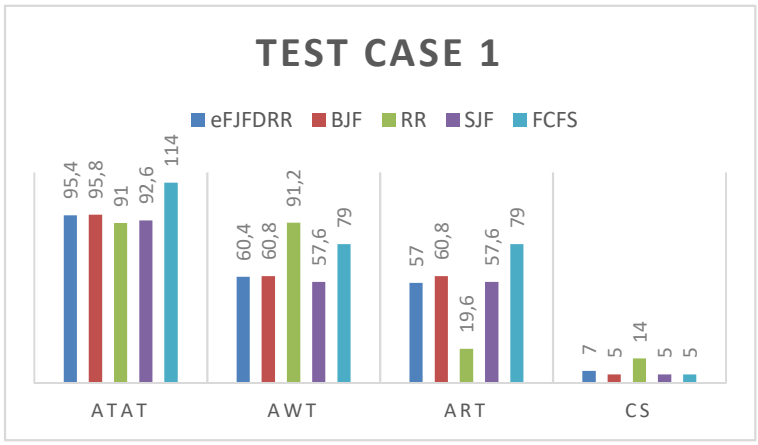


**Figure 1.** Comparative Results for Test Case 1

***Test Case 2:*** The researchers assume 5 processes. Processes with shorter burst times arrive earlier and have lower priority while processes with longer burst times arrive later and have higher priority. Table 4 shows the case parameters including the computed Fit Factor for each process. Table 5 shows the comparative results of eFJFDRR algorithm against the other algorithms assumed as benchmarks.

**TABLE 4.** eFJFDRR RESULT ON TEST CASE 2

| Process | Arrival Time | Priority | Burst Time | Fit Factor |
|---|---|---|---|---|
| P0 | 1 | 10 | 34 | 19.6 |
| P1 | 11 | 8 | 58 | 28 |
| P2 | 54 | 6 | 79 | 35.2 |
| P3 | 88 | 3 | 98 | 41 |
| P4 | 92 | 1 | 101 | 41 |

**TABLE 5.** eFJFDRR RESULT ON TEST CASE 2

| Algorithm | TQ | ATAT | AWT | ART | CS |
|---|---|---|---|---|---|
| eFJFDRR | 34, 58, 101 | 139.6 | 65.6 | 65.6 | 5 |
| BJF | N/A | 139 | 65 | 65 | 5 |
| RR | 15 | 223.4 | 198.6 | 22 | 27 |
| SJF | N/A | 139 | 65 | 65 | 5 |
| FCFS | N/A | 139 | 65 | 65 | 5 |

For all metrics, each algorithm performed roughly on equal ground in this test case except for RR which expectedly has excellent results in ART because of a constant time-quantum but has poor performance in the remaining metrics. Figure 2 below shows the comparative results of each algorithm.

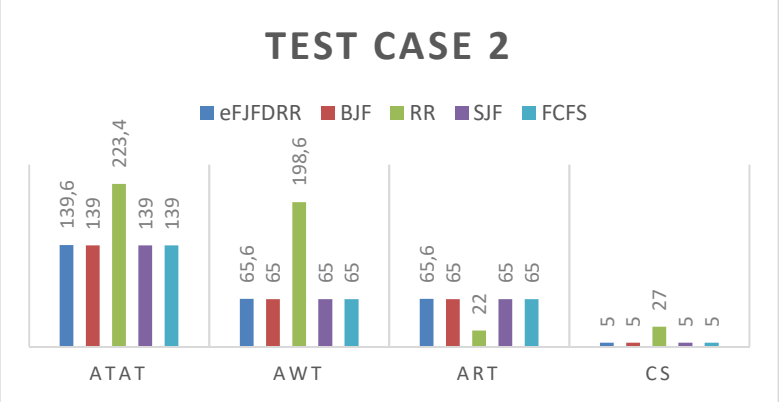


**Figure 2:** Comparative Results for Test Case 2

***Test Case 3:*** The researchers assume 5 processes. Processes that have higher priorities arrive earlier with random burst times. Table 6 presents the test case parameters including the computed Fit Factor for each process. Table 7 shows the comparative results of eFJFDRR against the algorithms assumed as benchmarks.

**TABLE 6.** TEST CASE 3

| Process | Arrival Time | Priority | Burst Time | Fit Factor |
|---|---|---|---|---|
| P0 | 1 | 1 | 80 | 32.6 |
| P1 | 3 | 1 | 29 | 12.2 |
| P2 | 20 | 1 | 40 | 16.6 |
| P3 | 31 | 5 | 17 | 9.8 |

| | | | | |
|---|---|---|---|---|
| P4 | 70 | 10 | 12 | 10.8 |

TABLE 7. eFJFDRR RESULT ON TEST CASE 3

| Algorithm | TQ | ATAT | AWT | ART | CS |
|---|---|---|---|---|---|
| eFJFDRR | 80,23,11,6 | 101 | 65.4 | 59.6 | 7 |
| BJF | N/A | 107.8 | 72.2 | 72.2 | 5 |
| RR | 15 | 113.8 | 103.2 | 23.8 | 14 |
| SJF | N/A | 95.4 | 59.8 | 59.8 | 5 |
| FCFS | N/A | 112.4 | 76.8 | 76.8 | 5 |

Similar to Test Case 1, eFJFDRR has the best ART in this test case, second only to RR. ATAT and AWT scores were also better than others excluding SJF. The CS score was average. Figure 3 shows the comparative results of each algorithm.

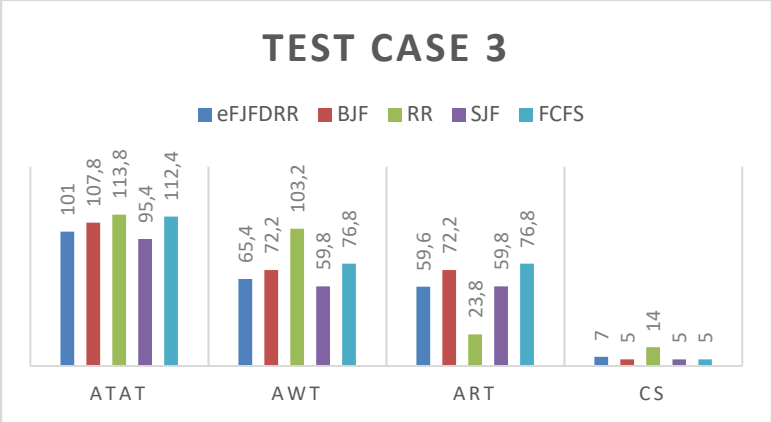


**Figure 3.** Comparative Results for Test Case 3

***Test Case 4:*** The researchers assume 5 processes. Processes that have longer burst times arrive earlier with random priorities. Table 8 presents the test case parameters including the computed Fit Factor for each process. Table 9 shows the comparative results of eFJFDRR against the other algorithms assumed as benchmarks.

**TABLE 8.** TEST CASE 4

| Process | Arrival Time | Priority | Burst Time | Fit Factor |
|---|---|---|---|---|
| P0 | 1 | 1 | 30 | 12.6 |
| P1 | 3 | 10 | 40 | 22 |
| P2 | 10 | 2 | 20 | 9.2 |
| P3 | 30 | 1 | 10 | 4.6 |
| P4 | 34 | 1 | 15 | 6.6 |

**TABLE 9.** eFJFDRR RESULT ON TEST CASE 4

| Algorithm | TQ | ATAT | AWT | ART | CS |
|---|---|---|---|---|---|
| eFJFDRR | 30, 20, 10, 20 | 55.4 | 32.4 | 30.4 | 6 |
| BJF | N/A | 51.4 | 28.4 | 28.4 | 5 |
| RR | 15 | 76.4 | 69 | 24.4 | 9 |
| SJF | N/A | 49.4 | 26.4 | 26.4 | 5 |
| FCFS | N/A | 67.4 | 44.4 | 44.4 | 5 |

This test case showed processes with higher fit factor scores arriving earlier. eFJFDRR performed closest to par with BJF and SJF in this test case. Figure 4 below shows the comparative results of each algorithm.

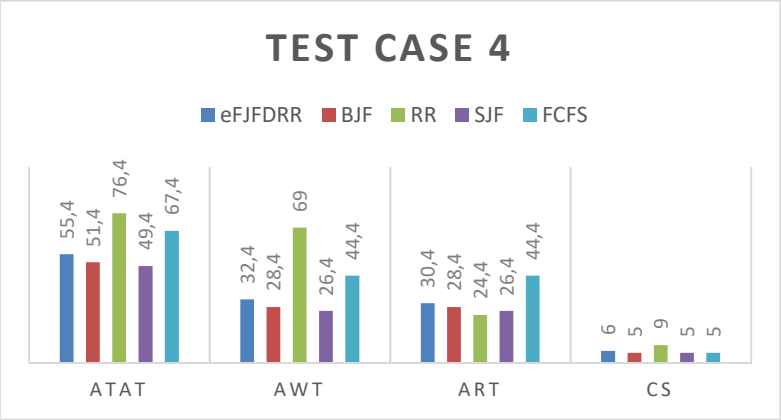


**Figure 4.** Comparative Results for Test Case 4

As the summary of the findings, we averaged the scores of each algorithm for all test cases. Results showed that eFJFDRR has reduced ATAT, AWT, and ART while maintaining a balanced CS. However, SJF yielded better results across all metrics as exemplified by figure 5.

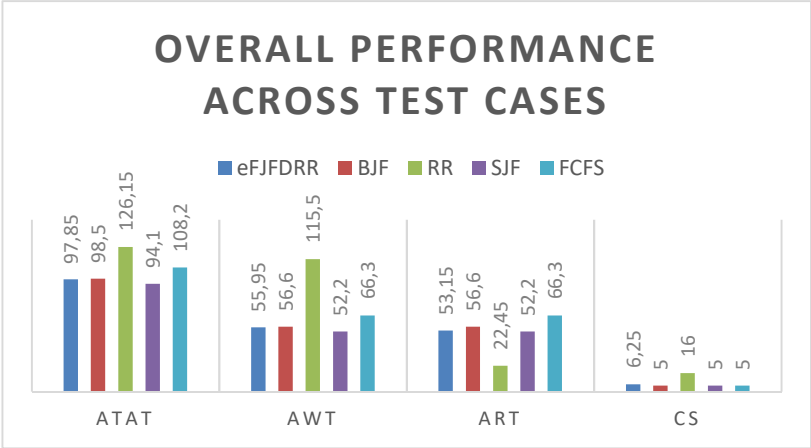


**Figure 5.** Comparative Results across all Test Cases

## 5. Discussions

Based on the Test Cases, the enhanced Fittest Job First Dynamic Round Robin (eFJFDRR) scheduling algorithm improves CPU scheduling on three metrics: the average turnaround time, average waiting time, and average response time. The average turnaround time and average waiting time improvements were evident in Test Cases 1 and 3, where the scores for both metrics were better than all algorithms except the shortest job first scheduling algorithm. This is despite the fact that processes with lower priorities arrived earlier in Test Case 1 and processes with higher priorities arrived earlier in Test Case 3. The researchers infer that the weight of priority in our algorithm is not as significant as it is in other algorithms like the best job first scheduling algorithm where due to it being non-preemptive and it schedules processes only using fit factor, priority is imperative. On the other hand, reduction on average response time was evident in Test Case 1, where our algorithm surpassed all others except round robin algorithm. However, we argue that round robin's excellent score in the metric can be offset by its poor performance in the remaining ones.

Results also show that eFJFDRR has the most balanced context switch score based on the number of processes submitted. The researchers attribute this to the dynamic computation of time quantum before a queue is executed; making our algorithm flexible enough to adjust throughput depending on a process' parameters. In contrast, the best job first, first come first serve, and shortest job first are all non-preemptive algorithms and round robin implements a static time quantum so flexibility is limited.

Furthermore, the eFJFDRR was observed to always be close to par with both best job first and shortest job first algorithms, sometimes surpassing them. In Cases 1 and 3, it was able to surpass best job first in three metrics: average turnaround time, average waiting time, and average response time as discussed above. It surpassed shortest job first only in one metric, the average response time, in Case 1.

While we succeeded in discovering individual test cases where our algorithm performed best, Figure 4 showed that it was only second to SJF in every metric when averaged across all test cases. The researchers surmise that this might be caused by the small difference in standard deviation of BJF and SJF on performance metrics [8]. As FJFDRR was originally based on BJF and our algorithm is a modification of FJFDRR, it is plausible that the experimental test cases we conducted were not sufficient in number and variety to distinguish why SJF performed better than eFJFDRR in this study.

## 6. Conclusions

Comparative results per test case showed that our algorithm can reduce average turnaround time and average waiting time. The researchers also discovered that the enhanced Fittest Job First Dynamic Round Robin scheduling algorithm is flexible enough to adjust throughput despite disparities in process' burst time and priority by maintaining a balanced context switch score. These results align with previous performance evaluations done with FJFDRR [10] where average turnaround time, average waiting time, and context switches were similarly reduced. In addition, our study revealed that the enhanced Fittest Job First Dynamic Round Robin scheduling algorithm can reduce average response time in some cases.

However, we also found that the number of our test cases and their varieties were not sufficient to properly distinguish why SJF performed better than eFJFDRR when averaged across all test cases. Thus, additional test cases are recommended.

## References

[1] Zahedi M H and Naghibzadeh M 2008 Fuzzy round robin cpu scheduling (FRRCS) algorithm *Adv. in Comp. and Information Sci. and Eng.* pp 348-53
[2] Helmy T and Dekdouk A 2007 Burst round robin as a proportional-share scheduling algorithm *IEEEGCC* (King Fahed University)
[3] Silberchatz A, Galvini P B and Gagne G 2013 *Operating Systems Concept* (Courier Kendallville) pp 261-6
[4] El-Sayed E, Rodriguez J M, Sushimita M and Thampi S 2016 *Intelligent Systems Technologies and Applications* (Springer International Publishing) p 318
[5] Fatai A, Tarek H and Sallam E 2010 An efficient randomized algorithm for real-time process scheduling *PicOS Operating Syst. Adv. Techniques in Computing Sci. and Software Eng.* p 120
[6] Xu C 2005 *Scalable and Secure Internet Services and Architecture* (Chapman & Hall/CRC) p 62
[7] Reddy C M 2009 *Operating Systems Made Easy* (University Science Press) p 33
[8] Al-Husainy M 2007 Best-job-first CPU scheduling algorithm *Information Technol. J.* vol **6** no **2** pp 288-93
[9] Mohanty R, Das M, Lakshmi P and Sudhashree 2011 Design and performance evaluation of a new proposed fittest job first dynamic round robin (FJFDRR) scheduling algorithm *Int. J. of Comp. Information Sys.* vol **2** no **2**
[10] Matarneh R 2009 Self-Adjustment time quantum in round robin algorithm depending on burst time of the now running processes *American J. of Applied Sci.* vol **6** no **10** pp 1831-7