Improved Round Robin Using Dynamic Time Quantum

1st *Valter A. Machado*
University of the Cordilleras
Baguio City, Philippines
valterandremachadodinho@gmail.com

2nd *Amr A. Shamsan*
University of the Cordilleras
Baguio City, Philippines
dam.m14@yahoo.com

3rd *Claizer Agaser C.*
University of the Cordilleras
Baguio City, Philippines
claizerjay@gmail.com

Abstract*—There are n number of CPU Scheduling algorithms available. In this paper, we explored improvements on the RR by combining it with SJF algorithm and we used the arrival time of the shortest process in our queue to be our Time Quantum every single Round Robin round. We then compared the performance of the proposed algorithm called improved Round Robin with Shortest Job First (iRRSJF) as with the RR algorithm together with the other CPU scheduling algorithms. Trial results showed that iRRSJF scheduling algorithm performed better in reducing average waiting time and average turnaround time than RR. All experimental test cases were performed with the following assumptions: Processes are executed in a single processor. All the processes are CPU bound. In this research we will also tackle if the algorithm is effective by the utilization of automaton figures and methods, whether the algorithm can cause benefits or a downfall when it comes to sorting algorithms. All parameters, namely total number of processes, arrival time and burst time for each process are known before CPU execution starts. FCFS, SJF and RR are used as benchmark algorithms. Our algorithm relies strongly on the arrival time of the shortest burst time. The researchers also discovered that some cases our algorithm is better than RR, SJF, and FCFS. However, we also found that the number of our test cases and their varieties were not sufficient to properly distinguish why SJF and FCFS performed better than iRRSJF when averaged.

Keywords —Q: Queue; P: Process; RQ: Ready Queue; BT: Burst Time; SJF: Shortest Job First; RR: Round Robin; SP: Shortest Process; SPAT: Shortest Process's Arrival Time; f = TQ == SPAT

I. INTRODUCTION

Scheduling is likewise an essential capacity of an operating system. Practically all computer devices and system are scheduled before use. As CPU is one of the primary computer resources. therefore, its scheduling is central to O.S designs. Scheduling is the strategy by which the system decides which task should be executed at any given time. Whenever the CPU became ideal the O.S must select one of the processes in the ready queue to be executed. An operating system is a software act as an intermediary between the user of a computer and computer hardware. The purpose of an O.S is to provide an environment in which a user can execute programs in an efficient manner. An operating system provided functionality is device management, memory management, file management, CPU Scheduling, process management, protection and security.

II. REVIEW OF RELATED LITERATURE

Round Robin becomes one of the most generally utilized scheduling disciplines regardless of its serious issue which rose because of the idea of a fixed predetermined time quantum. Since RR is used in almost every operating system (Windows, BSD, UNIX and Unix- based etc...), many researchers have tried to fill this gap, but still much less than needs. Helmy Et Al. propose a new weighting technique for Round-Robin CPU scheduling algorithm, as an attempt to combine the low scheduling overhead of round robin algorithms and favor short jobs. Higher process weights mean moderately higher time quantum; shorter jobs will be given additional time, with the goal, they will be removed earlier from the ready queue. Other works have utilized mathematical methodologies, giving new procedures using mathematical theorem.

Matarneh found that an ideal time quantum could be determined by the middle of burst times for the arrangement of processes in ready queue, except if this median is less than 25ms. In such case, the quantum value must be modified to 25ms to avoid the overhead of context switch time. Other works, have also used the median approach, and have obtained good results.

III. METHODOLOGY

Methodology for iRRSJF are the following:

- Fundamentally what we will do with our proposed algorithm, is arranging our processes by shortest job, then assign the arrival time (AT) of our current process in the ready queue as our time quantum (TQ) and then change the TQ every Round Robin round by assigning always the current process's AT as the TQ.

- First off, the algorithm will sort the ready queue by their shortest job (shortest BT), then set the TQ as the AT of the shortest process in the sorted ready queue (it will definitely be placed at the top of the ready queue). We will have a dynamic TQ based on the AT of the process with shortest BT (basically that will be our current process since the ready queue will always remain sorted by shortest job).

- The algorithm will work utilizing RR rounds if necessary, implying that if the first round has finished executing the algorithm will check if there is still processes with remaining BT, if yes it will go for a 2nd round assigning a new TQ. And it keeps looping round by round until it has no remaining BT.

3.1 iRRSJF Pseudocode

1. Let array be the ready queue.
2. enter AT and BT
3. Add the processes at the ready queue.
4. Sort the ready queue by SJF
5. Let TQ be set to SPAT (that is the current process because RQ is already sorted).
6. Execute the ready queue for 1 cycle round.
7. While (executing ready queue) {remove process with no remaining BT from the queue.}
8. If (there is still any process with remaining BT at the ready queue) {Check the shortest job among the processes, then execute the ready queue for one more round with the SPAT as your TQ.}
9. Else (no more remaining BT) {print (AWT and ATAT)}
10. Go to Step 1-9.

3.2 Illustrations

Finite-State Automata (FSAs), are used for modeling computation systems that can have changes in state. FSM systems have a finite number of states, and the flow of logic execution is dependent on the flow or transition between states. The transition from one state to another depends on the current state the machine is in, as well as the input into a state.

Finite Automata (DFA) was used in this research to illustrate the behavior of iRRSJF scheduling algorithm based on the pseudocode and methodology of the same research, see the illustrations down below.

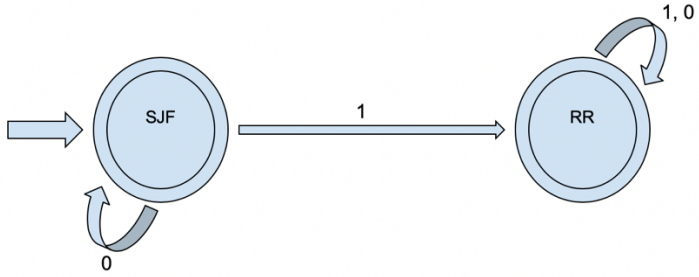
Key words:
Q: Set of states; Σ: Inputs (Alphabet); S: Start state/initial state; F: Set of final states; T: Transition function from Q x Σ -> Q

Let 0 be set to "no remaining BT" and 1 set to "remaining BT"
Q = {SJF, RR}
Σ = {0, 1}
S = SJF
F = {SJF, RR}
T =

iRRSJF DFA in table form:

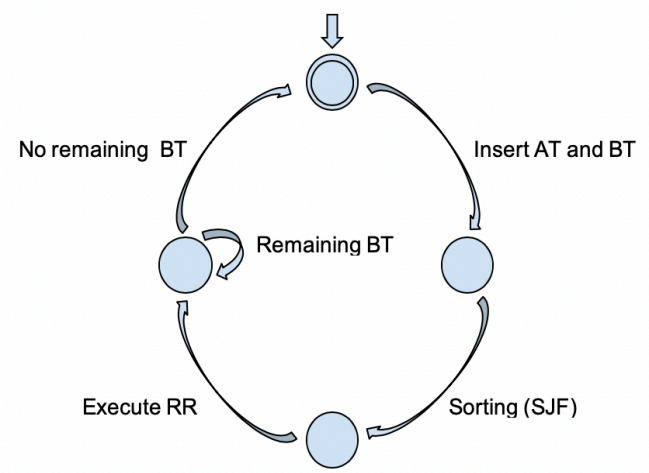
S	**F**	**Q**	**Σ**	
			0	**1**
→	*	SJF	SJF	RR
	*	RR	RR	RR

iRRSJF DFA in figure form:



- In this figure we can see two final states, the first final state role is to avoid ambiguity, just in case the algorithm will have only one process as its input (will have no need to execute the RR part), in other hand the second final state is the main final state because we expect to have more than one input.

iRRSJF Automaton:



- This automaton illustrates the behavior of the entire iRRSJF scheduling algorithm.

To further illustrate how the algorithm works, we present a simulation. Table 1 below shows 5 processes with arrival time and burst time all known before CPU execution.

TABLE 1:

Process No.	Arrival Time (AT)	Burst Time (BT)
P0	3	3
P1	1	2
P2	4	1
P3	7	6
P4	9	10

Assign TQ to AT of the current process in the sorted ready queue (P2 = 4), TQ = 4. Then execute the ready queue for 1 round, so in round 1 the ready queue will be the following: P2, P1, P0, P3 and

P4, since round 1 TQ = 4, processes P2, P1, and P0 Will be completed and removed from the ready queue, remaining in the ready queue P3=3 and P4=7. Then RR will go for another round to get the completion of the remaining processes. round 2, assign TQ to SPAT, TQ=7. Execute ready queue starting with SJF using TQ=7. When the round 2 is done in this specific case there will be no more processes to be executed.

IV. FINDINGS

All exploratory experiments were performed with the accompanying suppositions: Procedures are executed in a solitary processor. All the processes are CPU bound. All parameters, namely total number of processes, arrival time and burst time for each process, are known before CPU execution starts. FCFS, SJF and RR are used as benchmark algorithms. RR will have a time quantum set as the shortest process's arrival time every Round Robin round. The metrics used for algorithm comparison are the average turnaround time (ATAT), average waiting time (AWT). 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. All of these metrics are evaluated per algorithm with the lower scores signifying better performance. RR will have a TQ of 20.

4.1 Test Cases

Test Case 1: We assumed 5 processes. Processes with longer burst times arrive earlier and have shorter arrival time.

TABLE 2:

Process No.	Arrival Time (AT)	Burst Time (BT)
P0	1	80
P1	3	29
P2	20	40
P3	31	17
P4	70	12

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

Algorithm	TQ	AWT	ATAT
iRRSJF	70, 1	39.4	75
RR	20	84.4	120
SJF	N/A	59.8	95.4
FCFS	N/A	100.8	136.4

iRRSJF performed better than all the algorithms in ATAT and AWT in this test case as shown in Figure 1 below.

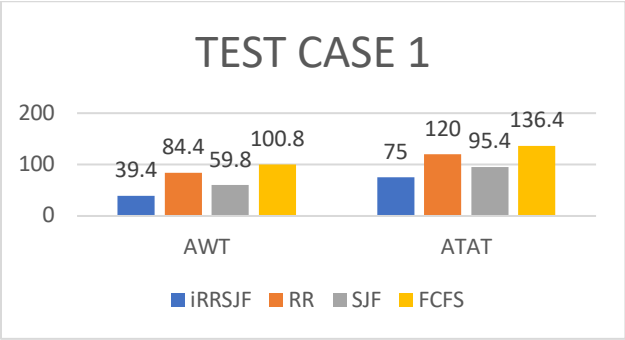


Figure 1. Comparative Results for Test Case 1.

Test Case 2: Let us say we have 5 processes, and the process with shortest BT has the lowest AT.

TABLE 4:

Process No.	Arrival Time (AT)	Burst Time (BT)
P0	1	22
P1	11	35
P2	54	52
P3	88	80
P4	92	77

TABLE 5 . RESULT ON TEST CASE 2

Algorithm	TQ	AWT	ATAT
iRRSJF	1, 11, 54	137.8	191.0
RR	20	131.4	184.6
SJF	N/A	26.6	79.8
FCFS	N/A	75.4	128.6

iRRSJF had a poor performance compared to all the algorithms in ATAT and AWT in this test case as shown in

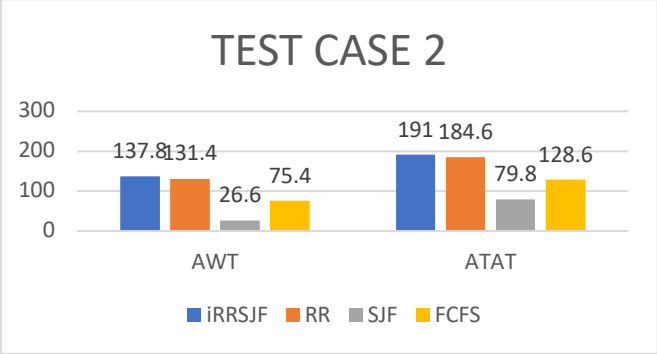


Figure 2 below.

Figure 2. Comparative Results for Test Case 2.

Test Case 3: Given 5 processes to be executed in different type of algorithms.
TABLE 6:

Process No.	Arrival Time (AT)	Burst Time (BT)
P0	54	34
P1	11	58
P2	17	79
P3	32	98
P4	70	101

TABLE 7 . RESULT ON TEST CASE 2

Algorithm	TQ	AWT	ATAT
iRRSJF	54, 11, 17, 32	188	262
RR	20	193.2	267.2
SJF	N/A	92.2	166.2
FCFS	N/A	113.2	187.2

iRRSJF had a poor performance compared to all the algorithms in ATAT and AWT, but had a fine performance in ATAT against RR in this test case as shown in Figure 3 below.

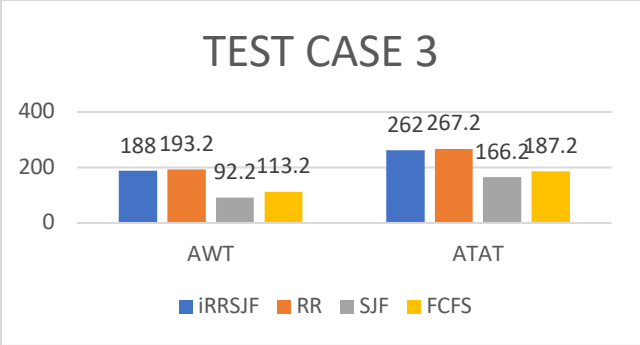


Figure 3. Comparative Results for Test Case 3.

Test Case 4: Given 5 processes, the one with the shortest BT has 4 as its AT, and the one with lowest AT has 20 as BT.

TABLE 8:

Process No.	Arrival Time (AT)	Burst Time (BT)
P0	1	20
P1	4	15
P2	10	10
P3	30	40
P4	34	30

TABLE 9. RESULT ON TEST CASE 4

Algorithm	TQ	AWT	ATAT
iRRSJF	10, 4, 1, 34	51	74
RR	20	41	64
SJF	N/A	19.2	42.2
FCFS	N/A	37	60

iRRSJF had a poor performance compared to all the algorithms in ATAT and AWT in this test case as shown in Figure 4 below.

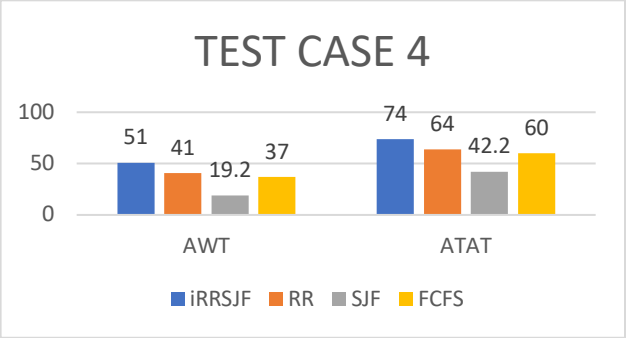


Figure 4. Comparative Results for Test Case 4.

As the summary of the findings, we averaged the scores of each algorithm for all test cases. Results showed that iRRSJF has reduced ATAT, AWT. However, SJF and FCFS yielded better results across all metrics, but our main goal was achieved, because our iRRSJF performed better than RR overall, as exemplified by figure 5.

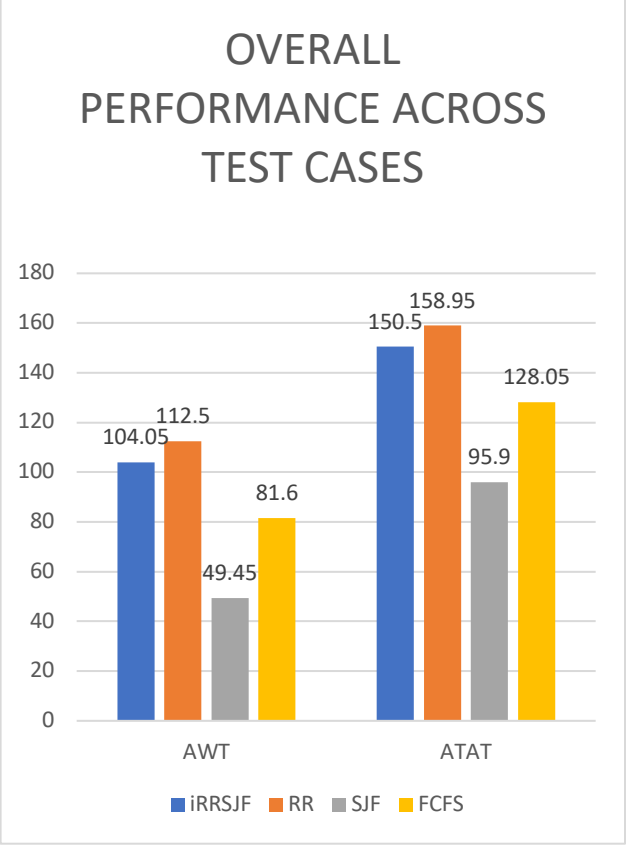


Figure 5. Comparative Results across all Test Cases

CONCLUSION

Comparative results per test case showed that our algorithm can reduce average turnaround time and average waiting time especially when it comes to compare to the existing RR algorithm, we have confirmed it in two of our four test cases. Our algorithms depends strongly on the arrival time of the briefest burst time. The researchers also discovered that at certain cases our algorithm is better than RR, SJF, and FCFS. However, we also found that the number of our test cases and their varieties were not sufficient to properly distinguish why SJF and FCFS performed better than iRRSJF when averaged. Additional test cases are recommended.

REFERENCES

[1] Silberschatz, A., P.B. Galyin and G. Gange, 2004, "Operating System Concepts". 7th Edn, John Wiley and Sons, USA, ISBN: 13:978-0471694663, PP: 944.

[2] Lingyun Yang, Jennifer M. Schopf and Ian Foster, "Conservative Scheduling: Using predictive variance to improve scheduling decisions in Dynamic Environments", SuperComputing 2003, November 15-21, Phoenix, AZ, USA.

[3] Silberschatz, Galvin and Gagne, Operating system concepts, 8th edition, Wiley, 2009.

[4] Weiming Tong, Jing Zhao, "Quantum Varying Deficit Round Robin Scheduling Over Priority Queues", International Conference on Computational Intelligence and Security. pp. 252-256, China, 2007.

[5] Jezreel Ian C. Manuel *et al* 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* 482 012046

[6] Shyam et al., International Journal of Advanced Research in Computer Science and Software Engineering 5(3), March - 2015, pp. 156-162

[7] IOP Conference Series: Materials Science and Engineering

[8] Armstrong Subero, Codeless Data Structures and Algorithms: Learn DSA Without Writing a Single Line of Code, 2020

[9] Neso Academy, Theory of Computation and Automata Theory, 2019

[10] Automata Theory, Warren McCulloch and Walter Pitts, 2004

[11] sciencedirect, Mohamed Hamada and Satoshi Sayota, 201

[12] Lesswrong, Ben Pace, 2019