Improved Global Routing Using A-Star Algorithm

AbdElrahman Elshafei*, Hossam Ahmed†, Mahmoud Mohamed‡ and Muhanad Atef§
Computer Engineering Dept., Faculty of Engineering, Cairo University
Cairo, Egypt
*abdelrahman.elshafei98@gmail.com, †hossamahmed201515@gmail.com, ‡mmmacmp@gmail.com, §muhanad.atef23@gmail.com

***Abstract*—In this paper we want to improve routing by improving global routing, this can be done by using A-Star algorithm, which will reduce time taken in this process and achieve the minimum wire length, many comparisons are taken in this paper with different algorithms to find the optimum algorithm to be used to achieve both minimum wire length and minimum time taken. From the comparisons of the paper we can find that using any algorithm is a trade off as when we decrease time taken, the wire length is increased and vice versa, so we cannot find an algorithm which is better from the other algorithms in general but we can say that using A-Star algorithm is a good approach to be used in global routing as it decreases the routing time however it is not the the best achieveable time but it is better than many other algorithms, and it is the best algorithm to achieve the minimum wire length with this complexity.**

***Index Terms*—VLSI Routing, Global Routing, Routing Algorithms, Fast Global Routing, Fast Routing Algorithms, Routing Algoritms Comparisons.**

I. Introduction

Routing is critical step in physical design process. Until now the optimum solution for VLSI routing has not been achieved yet, so it is considered as a very interesting challenging feild. It is exactly done in two steps, global routing and detailed routing. At first global routing is run which is the responsible for making an approximate routing for the whole circuit in order to be used as a guide for detailed routing, then the detailed routing is run to make the exact routing for the system. That means if global or detailed routing is improved the whole routing process is improved, but there are many problems we have to overcome to make a correct routing process. First of all we have to take into consideration the scale, as millions of wires exist in a small chip area which means that many kilometers of wires are placed in a very small area, so we have to minimize total wire length as much as we can, also we know that as the wire length increases the resistance increases as well which means more delay in the chip. We can find another problem as circuits are made in nano-scale which means that its geometric will be complex. Another problem that routing algorithm have to be applicable for more than one layer with different costs. We also have to take into consideration the direction of wires in every layer (vertical — horizontal) and no diagonal pathes, then to go from source 'S' to target 'T' the path taken should be in (vertical — horizontal) directions thet specified by the layer (at each layer wires are placed in one direction only), then there is another problem as when a wire goes from layer to another to continue on the perpendicular direction it have to go through via which have a high resistance. DFM (design for manufacturer) rules also has to be achieved. All of these constraints must be taken into consideration with the global routing to achieve hundred percent of the circuit connections, that means global routing will take a lot of time to achieve all these constraints, and here is the challenge to achieve all the routing specifications with minimum time taken.

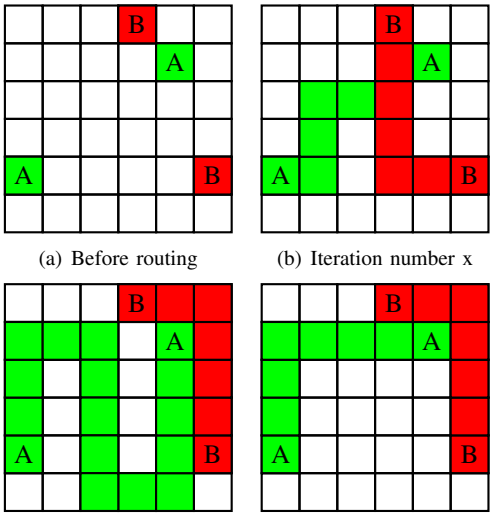


Fig. 1. Global routing to get the best pathes

Figure 1 shows a very simple approach of how the global router works. At (a) the source and target of both (A,B) need to be connected ignoring obstacles, nevertheless we can observe that the global router have to iterate to get the best routing pathes, at (b) (B,B) connected but there is no way to connect (A,A) as when (B,B) connected together they blocked the way for (A,A) to be connected, after some iterations we can find the figure at (c) in which (A,A) and (B,B) connected correctly, but there is a problem, the (A,A) connection is not the optimal path as there are pathes which achieve less wire length, so the global router have to iterate until reaching the optimal path. These iterations are done for only two connections in one layer without obstacles, then how about millions of wires in VLSI? this shows how much the global routing algorithm has to be very fast in order to connect this huge number of wires as fast as possible.

II. Related Work

Several papers proposed various types of approaches to improve the overall efficiency of the multilevel feedback queue scheduling algorithm. The chosen quantum time for each queue plays a major role. Hence, it is essential to choose a proper method to compute the time quantum value to minimize response time and maximize overall performance. In , an algorithm is introduced for minimizing the response time. In this algorithm, a Recurrent Neural Network (RNN) is used to determine both the number of queues and the optimized time quantum value for each queue. The RNN generates an effective model to compute the time quantum value. Their proposed intelligent version of the MLFQ offers good results, however, it suffers from a few drawbacks, the first being the direct proportionality of its network learning time and the amount of input data, and the second is the possibility of experiencing initial overhead at the first iterations of the algorithm. Our approach proposes an improved version of MLFQ that utilizes an altered version of RR named shortest remaining burst round-robin (SRBRR) introduced in which avoids the cost of learning time and overhead time in . Regarding the various approaches to improve the MLFQ algorithm, those attempts dealt with starvation by assigning different quantum values to the ready queues depending on their priority. Our approach also deals with starvation by boosting processes from lower priority queues to higher ones according to certain criteria.

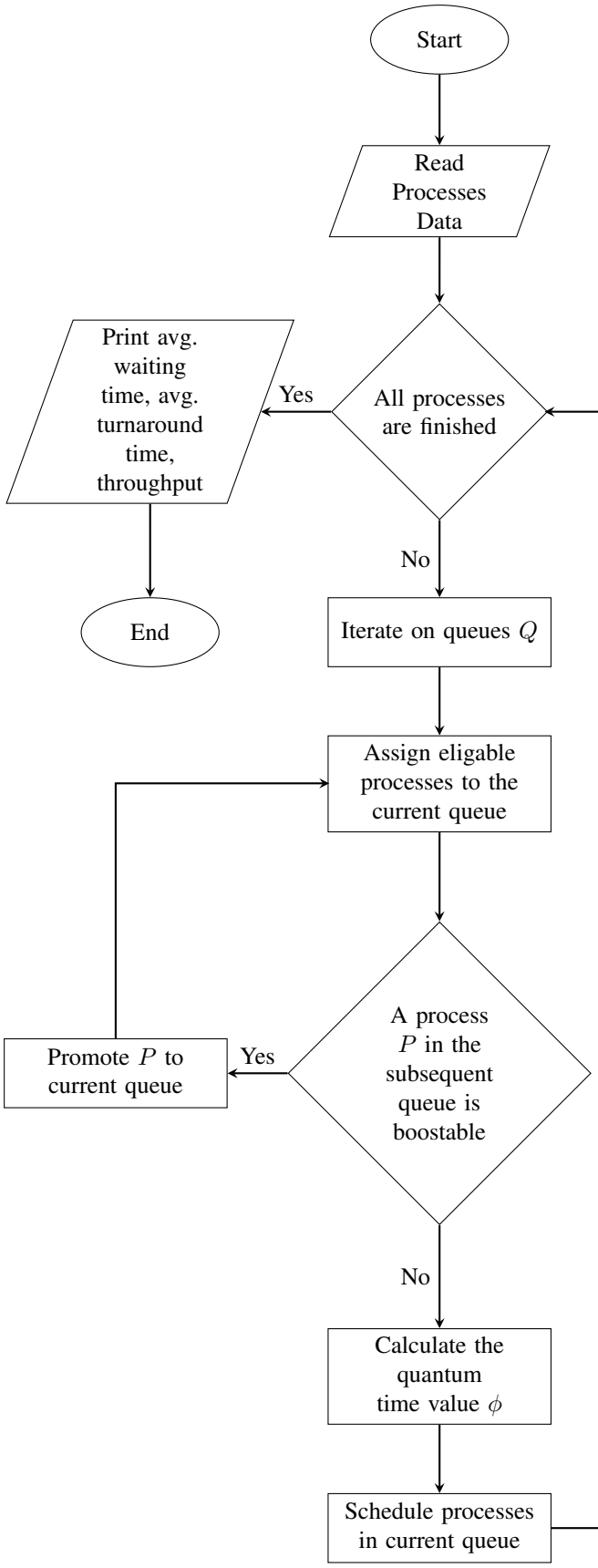
III. Proposed Approach

For a multilevel feedback queue scheduling algorithm, three parameters are considered. The first is the chosen scheduling algorithm for each queue, especially the last queue as its scheduling algorithm is expected to treat starvation. The second is the criteria according to which a process is promoted, this technique is also known as Aging. The third is the criteria according to which a process is demoted. In our proposed algorithm, there are 5 queues sorted in ascending order in line with their priority number, 1 being the highest priority and 5 being the lowest. Each queue uses a modified version of the round-robin scheduling algorithm stated in b3. In b3, processes are sorted in ascending order according to their burst time and are assigned a time quantum that equals the median burst time of those processes. This algorithm provides better turnaround time and waiting time than the standard static quantum RR algorithm whose quantum, if set too short, leads to many context switches and, if set too long, morphs the algorithm into an FCFS algorithm. The proposed alteration on the stated algorithm in b3 is that each queue quantum time equals the median burst time multiplied by a factor matching the queue number. Hence, the gradual increase of quantum time as priority decreases. For clarification, a queue with priority equal 2 has the following processes denoted by their burst time: 100, 300, 550, 600, 620, 700, 720, 900 and 1200, the median value is 620, since we are in a queue whose priority equals 2, therefore the quantum slice value according to the proposed approach equals 1240 as in $620 * 2$. If processes of a certain queue didn't terminate after assigning the quantum time value, they are shifted to the next lower priority queue. After introducing new processes into a queue, the quantum time slice is recalculated. Processes age whenever they are in a queue whose priority is one less than that currently getting scheduled and satisfies the following inequality:

$$\frac{waiting\ time\ of\ P}{burst\ time\ of\ P} >= 1 \tag{1}$$

Those procedures are repeated for all the generated queues until all processes reach the lowest priority queue where they are rescheduled until their completion.

A. Pseudocode Flowchart of the Proposed Approach



B. Proposed Algorithm

Algorithm 1 Developed Multilevel Feedback Queue Scheduling Algorithm

Input: Number of processes n, processes priority α burst time values β, arrival time values σ, context switching cost ϵ
Output: Average turnaround time, average waiting time, throughput

procedure DMLFQ
Declaration and Initialisation:
    Queue Q_i where $i \in \{1, 2, 3, 4, 5\}$
    Turnaround time values τ
    Finish time values λ
    Waiting time values θ
    Remaining time values $\mu = \beta$
    Quantum time value ϕ
    Number of context switches δ
    $time = 0$
    while $\exists P \in Q$ **do**
        for $i = 1$ to 5 **do**
            for $j = 1$ to n **do**
                if $\sigma_j <= time$ and α_j equals i **then**
                    Assign P_j to Q_i
                end if
                $waitingTime = time - \sigma_j$
                if $i < 5$ and $waitingTime/\mu_j >= 1$ and α_j equals $i + 1$ and $\sigma_j <= time$ **then**
                    Assign P_j to Q_i
                end if
            end for
            Sort Q_i in ascending order according to remaining time values
            $\phi = median\ value\ of\ Q_i * i$
            foreach $P_j \in Q_i$ **do**
                if $\mu_j <= \phi$ **then**
                    $time = time + \mu_j$
                    $\lambda_j = time$
                    Remove P_j from Q_i
                else
                    $time = time + \phi$
                    $\mu_j = \mu_j - \phi$
                    if $i < 5$ **then**
                        $\alpha_j = \alpha_j + 1$
                    end if
                end if
                if Previous process $\neq P_j$ **then**
                    $time = time + \epsilon$
                    $\delta = \delta + 1$
                end if
            end foreach
        end for
    end while
    for $i = 1$ to n **do** $\tau_j = \lambda_j - \sigma_j$, $\theta_j = \tau_j - \beta_j$
    end for
    return average of θ, average of τ, $n/time$
end procedure

IV. Experimental Analysis

A. Assumptions

The proposed scheduling algorithm is software simulated using a Python script which simulates scheduling independent CPU-bound processes on a single processor environment which guarantees that no more than a single process is getting scheduled at any arbitrary moment. Each process is assumed to have its own predetermined burst time, arrival time and the queue to which each one belongs. The proposed approach is non-preemptive. For the sake of giving an example, if a process was lately introduced to a queue denoted by Q_i prior to the current queue getting scheduled, it won't get scheduled until the current queue, its subsequent queues and the queues prior to Q_i get scheduled.

B. Experimental Scheme

On one hand, the input arguments to the proposed algorithm implementation are the number of processes to be scheduled, their burst time, their arrival time and the queue where each one belongs. On the other hand, output parameters are the average waiting time, average turnaround time and throughput. The following equations are used to calculate the previously mentioned output parameters:

$$Average\ Waiting\ Time = \frac{Total\ Waiting\ Time}{Number\ of\ Processes} \quad (2)$$

$$Average\ Turnaround\ Time = \frac{Total\ Turnaround\ Time}{Number\ of\ Processes} \quad (3)$$

$$Throughput = \frac{Number\ of\ Executed\ Processes}{Total\ Execution\ Time} \quad (4)$$

C. Performance Metrics

As a means to have a concrete, viable evaluation of either the proposed algorithm or any other scheduling algorithm, the output parameters are taken into consideration for analysis. Since the average waiting time indicates the average time that a process had to starve for, therefore the lower the average waiting time is the better. The same principle applies to the average turnaround time and the number of context switches, as the former implies the average time spent by the process since its arrival time to its completion and the latter costs time as the CPU is assigned back and forth between different processes. Contrarily to the prior metrics, the larger the throughput is the better as it indicates the number of processes that are completely executed per unit time.

D. Simulation

For the sake of showcasing the proposed algorithm, a number of processes, their predetermined burst time values and their arrival time values are taken as input to the Python simulation script. Suppose that the input to the script is according to the following table:

TABLE I

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	60	1
2	0	50	1
3	0	40	2
4	0	30	2
5	0	10	3
6	0	210	3
7	0	200	3

According to the proposed algorithm, the time quanta calculated are as follows:

TABLE II

Queue	*Quantum Value*
1	55
2	40
3	615
4	0
5	0

All processes are sorted in ascending order according to their remaining time and are scheduled by assigning the time quantum calculated for their respective queue. The time spent scheduling a particular queue is the waiting time for its subsequent queues.

The scheduling process goes as follows:

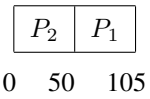


Fig. 2. Q_1 Gantt Chart

Considering that each process in Q_1 is assigned a quantum value of 50, as we reach the last process in Q_1, the total time elapsed equals 105, which happens to be the time that all the other processes in the subsequent queues had to wait for, hence the addition of their waiting time by a value of 105 units of time.

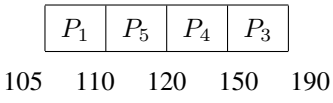


Fig. 3. Q_2 Gantt Chart

Even though P_5 is initially assigned to Q_3 as in Table I, it was promoted to Q_2 due to satisfying inequality (1).

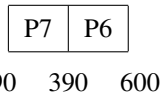


Fig. 4. Q_3 Gantt Chart

Whenever processes reach the lowest queue precompletion, they are scheduled using the RR scheduling algorithm with a relatively large quantum time value which is in most cases similar to using the FCFS algorithm because, as the time quantum value of an RR algorithm tends to infinity which could practically be a very large number relative to the available processes remaining time values, the algorithm tends to morph into the FCFS algorithm. This procedure is iterated until all the processes are finished. Simulation results are shown in the table below:

TABLE III

Avg. Turnaround Time	*Avg. Waiting Time*	*Throughput*
230	144.3	0.011667

E. Performance Comparisons

To assess the performance of the proposed algorithm implementation, multiple test cases are addressed and analyzed in seven different experiments. In each experiment, the output of the proposed algorithm implementation is compared to the output of another scheduling algorithm implementation addressed in a different paper, such as standard MLFQ algorithm with static quantum RR and other variants of MLFQ algorithms and RR algorithms.

1) Experiment 1: In this experiment, the proposed algorithm is compared against two MLFQ algorithm variants stated in b4. The first uses a static version of the RR algorithm for scheduling each queue, while the second variant uses a dynamic version of the RR algorithm for doing so.

TABLE IV
EXPERIMENT 1 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	1	25	1
2	5	70	1
3	6	84	1
4	7	17	1
5	8	35	1

TABLE V
EXPERIMENT 1 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	115	68.8
Dynamic RR MLFQ	150.8	107.6
Static RR MLFQ	161.4	116.2

2) Experiment 2: In this experiment, the proposed algorithm is compared against two MLFQ algorithm variants stated in b4. The first uses a static version of the SJFRR algorithm for scheduling each queue, while the second variant uses a dynamic version of the SJFRR algorithm for doing so.

TABLE VI
EXPERIMENT 2 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	1	25	1
2	5	70	1
3	6	84	1
4	7	17	1
5	8	35	1

TABLE VII
EXPERIMENT 2 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	115	68.8
Dyn. SJFRR MLFQ	134	91.8
Stat. SJFRRMLFQ	143.4	98.2

3) Experiment 3: In this experiment, the proposed algorithm is compared against two MLFQ algorithm variants stated in b5. The first uses a static version of the SJFRR algorithm for scheduling each queue, while the second variant uses a dynamic version of the SJFRR algorithm for doing so.

TABLE VIII
EXPERIMENT 3 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	8	1
2	3	133	3
3	2	21	2
4	8	39	2
5	19	67	2
6	33	114	3
7	33	54	2

TABLE IX
EXPERIMENT 3 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	151	88.7
Dyn. SJFRR MLFQ	252	119
Stat. SJFRR MLFQ	351	228

4) Experiment 4: In this experiment, the proposed algorithm is compared against multiple variants of the MLFQ

algorithm that are stated in b6: standard MLFQ algorithm, a priority-based MLFQ algorithm and a vague logic-based MLFQ algorithm.

TABLE X
EXPERIMENT 4 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	40	1
2	0	30	1
3	0	50	1
4	2	70	1
5	4	25	1
6	6	60	1
7	7	45	1

TABLE XI
EXPERIMENT 4 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	185.85	140.14
VMLFQ	190	170
MLFQ	232.14	175
PMLFQ	240	180

5) Experiment 5: This experiment is the same as the previous one, but with a different input test case.

TABLE XII
EXPERIMENT 5 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	90	1
2	0	30	1
3	0	28	1
4	0	57	1
5	0	73	1
6	0	19	1
7	0	42	1
8	0	67	1

TABLE XIII
EXPERIMENT 5 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	212.5	161.75
VMLFQ	260	225
MLFQ	290	240
PMLFQ	300	245

6) Experiment 6: In this experiment, the proposed algorithm is compared against two variants of the RR algorithm stated in b3. The first is a static version of the RR algorithm with a constant quantum value of 25 for scheduling each queue while the second uses a dynamic version of the RR algorithm called SRBRR for doing so.

TABLE XIV
EXPERIMENT 6 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	13	1
2	0	35	1
3	0	46	1
4	0	63	1
5	0	97	1

TABLE XV
EXPERIMENT 6 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	113.2	62.4
Dynamic SRBRR	122.4	71.6
Static RR	148.2	97.4

7) Experiment 7: This experiment is the same as the previous one, but with a different input test case. Note that for this test case, a process queue number is irrelevant to both the RR algorithm and the SRBRR algorithm mentioned in b3.

TABLE XVI
EXPERIMENT 7 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	54	1
2	0	99	3
3	0	5	2
4	0	27	2
5	0	32	2

TABLE XVII
EXPERIMENT 7 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Dynamic SRBRR	93.6	50.2
Proposed Algorithm	106.8	63.4
Static RR	152.2	108.8

F. Observation

From the above simulations of different test cases and multiple performance comparisons that involved as many as 11 different scheduling algorithms not including this paper

algorithm, it is clear that the average turnaround time and the average waiting time of the proposed algorithm is less than or – in few occasions – nearly equal to those of the stated algorithms. With that said, the proposed algorithm is arguably advantageous over those algorithms, considering even the case in which it underperformed compared to the SRBRR algorithm, it is still favourable due to the capability to separate processes into categories based on their need for the processor and other advantages of the MLFQ algorithm. The performance of the proposed algorithm compared to other algorithms is further illustrated in the following graphs:

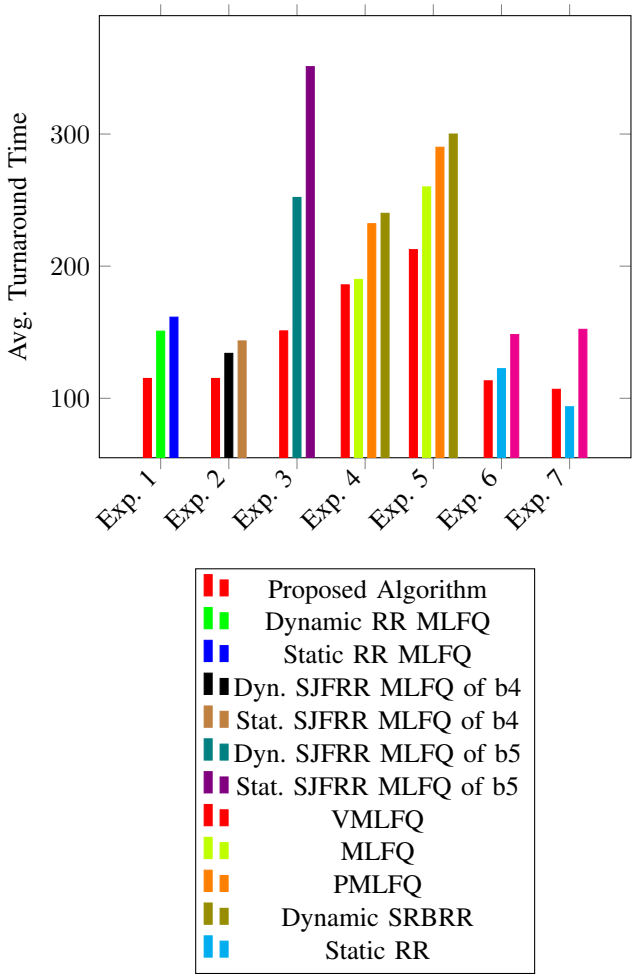


Fig. 5. Comparison graph for average turnaround time

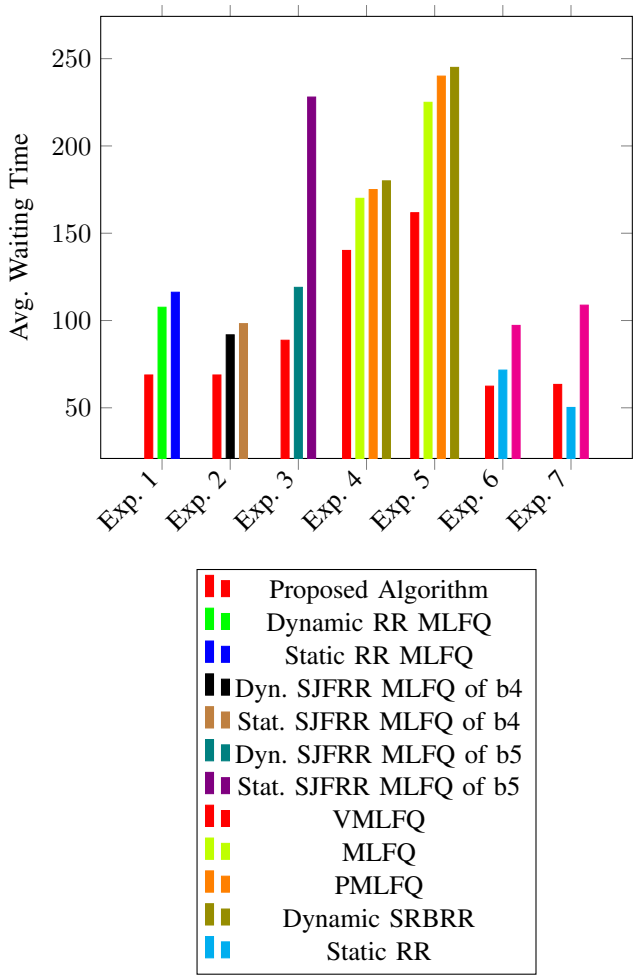


Fig. 6. Comparison graph for average waiting time

V. Conclusion and Future Work

The goal of this paper is to tackle different shortcomings associated with the standard MLFQ scheduling algorithm as well as its variants discussed in several papers. To resolve these deficiencies, we introduced different adjustable policies and techniques. It is evidently clear that those methods yield better CPU performance and optimize utilization by reducing the average waiting time as well as the average turnaround time. Despite experimenting numerous combinations of the parameters and scheduling policies by which the proposed MLFQ algorithm operates, we can say that there is yet a large room for experiment and improvement through finding better methods and policies which would make the algorithm more adaptable to the nature of the submitted processes and overall more enhanced. For instance, adjusting the criteria by which the schedular decides whether to promote a process might further lessen starvation. Making the proposed algorithm preemptive might also aid mitigating starvation and render overall better performance. The possibilities are limitless.

References

[1] Muhammet Mustafa Ozdal and M. D. F. Wong, "Archer: a history-driven global routing algorithm," *2007 IEEE/ACM International Conference*

on Computer-Aided Design, San Jose, CA, 2007, pp. 488-495, doi: 10.1109/ICCAD.2007.4397312.