AMA488 Simulation

Mini-project Report

CHENG Yu (Chester) 14111621D

December 13, 2018

Contents

1 Introduction

The problem faced here is to determine the optimal machine numbers for each of the four stages of cotton production. From raw cotton, it required four stages which take place in sequence to complete the final order, which are spinning, weaving, finishing and packaging. The processes are all done by machines and the processing time for different stages follows different distributions. The management realized that if the factory is producing at its most efficient level, the overall percentage of idle time of the machines has to be minimized and also, it has decided that the maximum number of machines at each stage is 10. Therefore, they want to know the number of machines, subject to this constraint, that the factory should use at each stage in order to minimize the idle time. Since minimizing idleness is not equivalent to minimizing the queuing cost, the management also wish to know the queuing cost under optimal solution.

This problem can be solved by computer simulation. This report will firstly formulate the problem in mathematical way, then provide some preliminary simulation knowledge such as how to generate random variables follow various distributions. Two simulation approach will be discussed with focus on an event-based simulation method and detailed algorithms will be given for it. The results will also be analyzed after the simulation and this essay will be concluded by some discussion on future improvement. All simulations in this project are done in R and the full R code will be attached in the appendix.

2 Problem formulation

As given in the problem, the time horizon is one week and all of the time is in second ($7 * 24 * 3600 = 604800s$). Denote the number of of machines in spinning, weaving, finishing and packing as N_A, N_B, N_C, N_D and time to process in each of the four stages as T_A, T_B, T_C, T_D. The distributions of processing time are:

$$T_A \sim \mathcal{N}(240, 120)$$

$$T_B \sim \mathcal{N}(480, 200)$$

$$T_C \sim exp(\frac{1}{120})$$

$$T_D \sim exp(\frac{1}{360})$$

Note that $1 \leq N_A, N_B, N_C, N_D \leq 10, N_A, N_B, N_C, N_D \in \mathbb{N}$.

Initially, the queue length for machine A is 25 and when a final product is produced, the queue length will increase by 1. For the queues in weaving, finishing and packing processes, there is no length limit of the queues. Products in the queues will be randomly chose to process when available machines comes up, e.g. if there are 10 products in the queue and 2 machines are available, 2 products will be selected randomly regardless to the waiting time of each product.

For a machine, it can be in only two status, "work" or "idle". Therefore, we have the following equation for each machine:

$$total\,time = work\,time + idle\,time \tag{2.1}$$

Therefore, to minimize the overall idle time percentage, we can only focus on calculating the total work time for each machine.

3 Preliminaries

Before introducing the simulation approach deployed, it's essential to have some preliminaries on simulation. This section will first introduce the test of uniformity and then illustrate how to generate normal and exponential random variables from the uniform random variable.

The most important part of simulation is generating random number, which makes simulation meaningful. From random numbers, we can easily generate the uniform distribution random variable which can be used in generating other random variables. However, are the uniform random variables generated really uniform? Therefore, some statistical tests need to be carried out first.

3.1 Test for uniformity

K-S test is widely used in testing for uniformity in statistics. We first generate 100 random variables uniformly distributed over (0,1) and implement K-S test to see if the uniform random variable generator is valid. According to what learned from class, the function KS_test can be implemented as follows:

```
# Function for K-S test of random number generator
# by default use 95% confidence level, N is large
KS_test<-function(N,alpha=0.05){
  test_u=runif(N,min=0,max=1)
  sort_u=sort(test_u)
  i=1:N
```

```
  pct_1=i/N
  pct_2=(i-1)/N
  diff_1=pct_1-sort_u
  diff_2=sort_u-pct_2
  D_star=max(max(diff_1),max(diff_2))
  D_critical=1.36/sqrt(N)
  if (D_star<D_critical){
    result="Pass"
  }
  else{
    result="Fail"
  }
  return(result)
}
```

The result for this function actually is "Pass", meaning that the generated random variate are uniformly distributed over (0,1).

3.2 Generate random variables

From the above section, we know that we have a (0,1) uniform random variable generator. Now, let's focus on how to generate normal random variables and exponential random variables. The exponential random variable can be generated from by inverse transformation method and the standard normal random variable can be generated by polar method. Assuming that we have a random number generator that can generate random number $u \sim Unif(0,1)$. The above methods can be implemented in R as follows:

Exponential random variable:

```
# Function for generating exponential random variable
exp_gen<-function(mean_time){
  u=runif(1)
  x=(-mean_time)*log(1-u)
  return(x)
}
```

Standard norml random variable:

```
# Funtion for generating standard normal random variables
std_norm_gen<-function(){
  u_1=runif(1,min = 0, max = 1)
```

```
  u_2=runif(1,min = 0, max = 1)
  v_1=2*u_1-1
  v_2=2*u_2-1
  w=v_1^2+v_2^2
  while(w>1){
    u_1=runif(1,min = 0, max = 1)
    u_2=runif(1,min = 0, max = 1)
    v_1=2*u_1-1
    v_2=2*u_2-1
    w=v_1^2+v_2^2
  }
  z_1=v_1*sqrt(-2*log(w)/w)
  z_2=v_2*sqrt(-2*log(w)/w)
  z<-c(z_1,z_2)
  return(z)
}
```

Then it can be transformed as:

$$x = \mu + z\sigma$$

where x is a normal random variate generated with mean mu and variance σ^2.

4 Simulation methodology

There are two approaches to simulate this problem, namely event-based simulation and time-based simulation. The basic concepts behind event-based simulation is that we are only concerned about the "key event", which has significant to the system status change. While the time-based simulation tries to check at every time point whether there is system status change. In this problem, event-based simulation seems more reasonable. Of course we can adapt time-based simulation by checking the system status, i.e. the status of each machine (work or idle). However, it seems not computational efficient and the rounding lower the accuracy. Therefore, this report will illustrate the event-based simulation.

4.1 Methodology overview

As stated above, in an event-based simulation, the focus is "key event". In this problem, since the target is minimizing overall percentage of idle time, intuitively, the key event is the change of each machine's status.

For each process, if there is enough available machines, the products in the queues will be processed. When a machine completes the process, this product will be passed into next stage or output as final product and the queue in the spinning process will increase by 1, which are equivalent to the queue in next stage increases by 1(from queues' sense, we can say that spinning is the next stage of packing). Therefore, to be more specifically, the key event that we need to consider is the process completion event.

We can record all of the key events' time in the clock and also the work time of a machine. The clock runs from time 0 to the end time, which is 604800s. Therefore, what we care most actually are two things:

- Which machine will complete its process earliest in the whole system?
- How long does it take to complete from now?

The above two questions will be answered by two algorithms shown as follows, which include firstly, how a product can be processed inside a stage so that we can know which machine inside a stage can complete earliest and secondly, how stages can interacts with each other so that we know among all earliest completed machines from different stages, which one will be the earliest in the whole system.

Therefore, we should be able to find for every machine, how long it works and therefore find the average percentage of idleness of each machine. The optimal solution can then be obtained by looping different combinations of the machine numbers in the four stages.

4.2 Algorithm 1: How each stage processes the product

Inside one stage, the structures for different stages are quite identical. The purpose for this algorithm is to find the earliest machine inside the stage that completes the process. This processing time will be a candidate of the earliest machine that completes a process in the whole system. Let Q denote the queue length of this stage and N denote the number of machines in this stage. As mentioned above, there are only two possible status of the machine and we define S as the status of the machine as follows:

$$S := \begin{cases} 1, & \text{if the machine is working} \\ 0, & \text{if the machine is idle} \end{cases}$$

For key event, actually, we are interested in the next earliest completion of the process for a machine, therefore, we also need record the remaining time for each machine in this stage if it is working. The total working time of the machine need to change correspondingly. The processing time for this stage need to be recorded and the clock time will be also be recorded so that all stages change simultaneously. Last but not least, we need to know which machine will be the released earliest. Therefore, for every stage, we can define a stage status table shown as follows (this is the initial situation, assume no queue):

Table 1: Stage status table

No. of machine	S	Remaining time	Total working time	total clock time	Q	Process time	Machine release
1	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	0	0	0	0	0	0	0

Actually, the contents for column 5-8 are identical for different machines within the same stage. Initially, Q are all set to be 0 apart from the spinning stage, which is 25 as given by the problem.

For any stage, we first check that whether there is queue and available machines. If yes, we decrease the queue length by the minimum between queue length and number of machines available. The minimum is the number of machines that going to be working. Then, generate processing time needed for every machine going to be working and update the third column of the table accordingly. Choose the non-zero minimum of the third column to be the value of the seventh column, which is the process time. Record its machine number in the last column. Note that this is just a candidate for the process time for the earliest completion event of the whole system.

If there is no queue or no machines available, then we just need to get the non-zero minimum of the third column, record it in the seventh column and record the machine number in the last column.

4.3 Algorithm 2: How stages can be linked together

After obtaining the candidates of the earliest process completion from each stage, we now need to consider the earliest completion of all machines from different stages. We first create an array to store all the shortest process time from different stages and find which one has the non-zero minimum process time. Take this one as the real process time to next earliest completion and record the number of stage it is. Then, what we need to do is to update all of the stage status matrices. Here are the rules:

- **For the earliest completed stage**, we first reduce the third column, which is the remaining time, by the real process time if the machine is working. Secondly, the forth column, which is the total working time, will have the real process time added. Next, total clock time will increase by the real process time. Finally, we can change the status of the machine will complete earliest to 0.

- **For the stage follows the earliest completed stage**, we need to we first reduce the third column, which is the remaining time, by the real process time if the machine is working. Secondly, the forth column, which is the total working time, will have the real process time added. Next, total clock time will increase by the real process time. Finally, we need to add 1 to the sixth column, since there is a product completed from the last stage and will go into the queue of this stage.

- **For the other stages**, we need to we first reduce the third column, which is the remaining time, by the real process time if the machine is working. Secondly, the forth column, which is the total working time, will have the real process time added. Finally, total clock time will increase by the real process time.

Also, the queue length for the spinning stage is limited to 25, which means that the queue may not increase until the next earliest completion occurs in stage 1 if the queue length of spinning stage reaches 25.

4.4 Connect with clock time and optimization

From 4.2 and 4.3, we can simulate the process once given some initial input of the status matrices. Note that all the clock time has been recorded in the status matrices, therefore, we can create a loop use this clock time. The full simulation can be carried as follows:

Step 1 Initialize 4 stage status matrices as shown in table 1, Queue length for spinning stage matrix is set to be 25.

Step 2 Set the clock time to be 0 and set an end time as 604800s.

Step 3 Do algorithm 1 and algorithm 2 (in actual situation, algorithm 1 is implemented into algorithm 2), set the clock time to be the clock time in the stage status matrices.

Step 4 If the clock time is smaller than the end time, then repeat Step 3 and if not, end the simulation and output the latest stage status matrices.

Step 5 Calculate the average working time percentage as the fraction of total working time for all machines and total clock time for all machines. Use 1-average working time percentage as the overall idle percentage.

We can repeat the above process for different combinations of machine numbers in different stages and store the corresponding overall idle percentage. Then find the lowest one, which is the optimal solution.

5 Implementation and results

According to the methodology illustrated the above section, the simulation is implemented in R.

5.1 Functions implemented

As seen from the algorithms above, the non-zero minimum is used quite often and there is no such build-in function in R, therefore, I created one function called "$special_min$" so that it can return the non-zero minimum of a vector if this vector does not consist of all 0 and return 0 if all of its elements are 0. The R code is shown as follows:

```
#### special minimum function ####
min_special<-function(x){
  if(sum(x)>0){
    result<-min(x[x>0])
  }
  else{
    result<-0
  }
  return(result)
}
```

Then the algorithm 1 is implemented individually for the four stages, here I just show the example of the spinning process:

```
#### function for spinning process ####
# N: number of spinning machines
# Q: queue length
# processing time ~ N(240,120)
# status: 0 for idle, 1 for busy
spin<-function(spin_machine){
```

```
N<-length(spin_machine[,1])
avail<-N-sum(spin_machine[,1])
Q<-spin_machine[1,5]

if(Q>0 & avail>0){
if(avail<=Q){
j=1
for(i in 1:N){
  if(spin_machine[i,1]==0 & j<=avail){
    spin_machine[i,1]=1
    spin_machine[i,2]<-rnorm(1,240,sqrt(120))
    j=j+1
  }
}
spin_machine[,5]<-spin_machine[,5]-avail
}
else{
j=1
  for(i in 1:N){
    if(spin_machine[i,1]==0 & j<=Q){
      spin_machine[i,1]=1
      spin_machine[i,2]<-rnorm(1,240,sqrt(120))
      j=j+1
    }
  }
spin_machine[,5]<-spin_machine[,5]-Q
}
t_process<-min_special(spin_machine[,2])
k<-match(t_process,spin_machine[,2])
spin_machine_finish<-spin_machine
spin_machine_finish[,6]<-rep(1,N)*t_process
spin_machine_finish[,7]<-rep(k,N)
}

else{
  t_process<-min_special(spin_machine[,2])
  k<-match(t_process,spin_machine[,2])
  spin_machine_finish<-spin_machine
  spin_machine_finish[,6]<-rep(1,N)*t_process
  spin_machine_finish[,7]<-rep(k,N)
  }
```

```

return(spin_machine_finish)
}
```

Therefore, there are four functions, namely, "spin", "weave", "finish" and "pack".

Then the algorithm 2 is implemented based on the the above 4 functions, just achieved as the described in 4.3 with name of "pass":

```
pass<-function(spin_machine,weave_machine,finish_machine,pack_
    machine){
  # check all process first
  spin_machine_check<-spin(spin_machine)
  weave_machine_check<-weave(weave_machine)
  finish_machine_check<-finish(finish_machine)
  pack_machine_check<-pack(pack_machine)
  # find the earliest finish time
  process<-c(spin_machine_check[1,6],weave_machine_check[1,6],
      finish_machine_check[1,6],pack_machine_check[1,6])
  real_process_time<-min_special(process)
  #find the earliest finish process
  real_process<-match(real_process_time,process)
  if(real_process==1){
    # release one machine in spin process
    spin_machine_out<-spin_machine_check
    spin_machine_out[,2]<-spin_machine_check[,2]-real_process_time*
        spin_machine_check[,1]
    spin_machine_out[,3]<-spin_machine_check[,3]+real_process_time*
        spin_machine_check[,1]
    spin_machine_out[,4]<-spin_machine_check[,4]+real_process_time
    spin_machine_out[spin_machine_out[1,7],1]<-0
    # add 1 to queue of weave process
    weave_machine_out<-weave_machine_check
    weave_machine_out[,2]<-weave_machine_check[,2]-real_process_
        time*weave_machine_check[,1]
    weave_machine_out[,3]<-weave_machine_check[,3]+real_process_
        time*weave_machine_check[,1]
    weave_machine_out[,4]<-weave_machine_check[,4]+real_process_
        time
    weave_machine_out[,5]<-weave_machine_check[,5]+1
    # let the time fly for finish process
    finish_machine_out<-finish_machine_check
```

```
27     finish_machine_out[,2]<-finish_machine_check[,2]-real_process_time*finish_machine_check[,1]
28     finish_machine_out[,3]<-finish_machine_check[,3]+real_process_time*finish_machine_check[,1]
29     finish_machine_out[,4]<-finish_machine_check[,4]+real_process_time
30     # let time fly for pack process
31     pack_machine_out<-pack_machine_check
32     pack_machine_out[,2]<-pack_machine_check[,2]-real_process_time*pack_machine_check[,1]
33     pack_machine_out[,3]<-pack_machine_check[,3]+real_process_time*pack_machine_check[,1]
34     pack_machine_out[,4]<-pack_machine_check[,4]+real_process_time
35   }
36   .
37   .
38   .
39   result<-list(spin_machine_out,weave_machine_out,finish_machine_out,pack_machine_out)
40   return(result)
41 }
```

Line 36-38 essentially repeat what line 12-35 does but in other case, which are 2,3,4, therefore, they are omitted and the full code can be checked in the appendix.

The final function is to conduct the simulation. The arguments are the machine numbers for each stage and the default argument is the initial queue length in spinning stage, which is set to be 25. The output is the overall idle percentage with the machine numbers. Detailed code of the "simulation" function is shown as follows:

```
simulation<-function(N_A,N_B,N_C,N_D,Q_0=25){
  spin_machine_begin<-matrix(0,N_A,7)
  weave_machine_begin<-matrix(0,N_B,7)
  finish_machine_begin<-matrix(0,N_C,7)
  pack_machine_begin<-matrix(0,N_D,7)
  spin_machine_begin[,5]<-Q_0*rep(1,N_A)
  while (t_clock<=t_end) {
    status<-pass(spin_machine_begin,weave_machine_begin,finish_machine_begin,pack_machine_begin)
    spin_machine_begin<-status[[1]]
    weave_machine_begin<-status[[2]]
    finish_machine_begin<-status[[3]]
```

```
    pack_machine_begin<-status[[4]]
    t_clock<-status[[1]][1,4]
  }
  occupy_rate<-sum(sum(status[[1]][,3]),sum(status[[2]][,3]),sum(
     status[[3]][,3]),sum(status[[4]][,3]))/sum(sum(status
     [[1]][,4]),sum(status[[2]][,4]),sum(status[[3]][,4]),sum(
     status[[4]][,4]))
  idle_rate<-1-occupy_rate
  result<-c(N_A,N_B,N_C,N_D,idle_rate)
  return(result)
}
```

5.2 Simulation and results

The "simulation" function is repeated 10000 (10^4) times and the corresponding idle percentages are recorded. The lowest 10 idle parentage' combinations are shown as follows:

Spinning	Weaving	Finishing	Packing	Avg_idle_rate
2	4	1	3	6.23%
3	6	2	5	7.98%
4	7	2	6	10.72%
4	8	2	6	11.06%
4	8	3	6	11.34%
4	8	3	7	11.36%
4	7	2	5	11.37%
3	5	2	4	11.69%
2	4	2	3	12.03%
3	6	2	6	12.24%

Figure 1: Top performance combinations

Therefore, we can see that the optimal combination is 2 machines in the spinning stage, 4 machines in the weaving stage, 1 machine in the finishing stage and 3 machines in the packing stage. The overall idle percentage is approximately 6.23%.

For the queuing cost, we can create one more column based on the table 1 to record the average queue length. For the average queue length, it is computaed as:

$$\text{original queue length} + \frac{\text{real process time} * \text{current queue length}}{\text{total time (604800)}}$$

real process time is column 7 and current queue length is column 6.

After inplemented in R, we have the following average queue length of each stage for the optimal solution:

Table 2: Average queue length for each stage

Spinning	**Weaving**	**Finishing**	**Packing**
4.37	0.28	4.79	6.26

Therefore, multipling the cost per stage we can find that the overall average queueing cost is

$$4.37 * 100 + 0.28 * 200 + 4.79 * 300 + 6.26 * 400 = 4434 \text{ (dollars/unit time)}$$

6 Conclusion

We can see that actually, the proportion for number of machines actually is closed to or equals to the proportion of mean processing time, i.e.

$$240 : 480 : 120 : 360 \Longrightarrow 2 : 4 : 1 : 3$$

There should be a theoretical method to analyze this tandem queuing model and analyze the steady state situation. This can be a future research point. Besides, this simulation method didn't consider the case that multiple machines may complete the services at the same time, which will probably bring differences in the results. Last but not least, the implementation coding can be improved since actually, some entrances of the status matrix are duplicated.

Appendix A Full R code for simulation

```
########## Simulation main program ##########

###### Part 1: Define several functions ######

#### special minimum function ####
min_special<-function(x){
  if(sum(x)>0){
    result<-min(x[x>0])
  }
  else{
    result<-0
  }
  return(result)
}

#### function for spinning process ####
# N: number of spinning machines
# Q: queue length
# processing time ~ N(240,120)
# status: 0 for idle, 1 for busy
spin<-function(spin_machine){
  N<-length(spin_machine[,1])
  avail<-N-sum(spin_machine[,1])
  Q<-spin_machine[1,5]

  if(Q>0 & avail>0){
  if(avail<=Q){
  j=1
  for(i in 1:N){
    if(spin_machine[i,1]==0 & j<=avail){
      spin_machine[i,1]=1
      spin_machine[i,2]<-rnorm(1,240,sqrt(120))
      j=j+1
    }
  }
  spin_machine[,5]<-spin_machine[,5]-avail
  }
  else{
  j=1
```

```
    for(i in 1:N){
      if(spin_machine[i,1]==0 & j<=Q){
        spin_machine[i,1]=1
        spin_machine[i,2]<-rnorm(1,240,sqrt(120))
        j=j+1
      }
    }
  spin_machine[,5]<-spin_machine[,5]-Q
  }
  t_process<-min_special(spin_machine[,2])
  k<-match(t_process,spin_machine[,2])
  spin_machine_finish<-spin_machine
  spin_machine_finish[,6]<-rep(1,N)*t_process
  spin_machine_finish[,7]<-rep(k,N)
  }

  else{
    t_process<-min_special(spin_machine[,2])
    k<-match(t_process,spin_machine[,2])
    spin_machine_finish<-spin_machine
    spin_machine_finish[,6]<-rep(1,N)*t_process
    spin_machine_finish[,7]<-rep(k,N)
    }

  return(spin_machine_finish)
  }

#### function for weaving process ####
# N: number of weaving machines
# Q: queue length
# processing time ~ N(480,200)
# status: 0 for idle, 1 for busy
weave<-function(weave_machine){
  N<-length(weave_machine[,1])
  avail<-N-sum(weave_machine[,1])
  Q<-weave_machine[1,5]

  if(Q>0 & avail>0){
  if(avail<=Q){
  j=1
    for(i in 1:N){
```

```
      if(weave_machine[i,1]==0 & j<=avail){
        weave_machine[i,1]=1
        weave_machine[i,2]<-rnorm(1,480,sqrt(200))
        j=j+1
      }
    }
    weave_machine[,5]<-weave_machine[,5]-avail
  }
  else{
  j=1
    for(i in 1:N){
      if(weave_machine[i,1]==0 & j<=Q){
        weave_machine[i,1]=1
        weave_machine[i,2]<-rnorm(1,480,sqrt(200))
        j=j+1
      }
    }
    weave_machine[,5]<-weave_machine[,5]-Q
  }
  t_process<-min_special(weave_machine[,2])
  k<-match(t_process,weave_machine[,2])
  weave_machine_finish<-weave_machine
  weave_machine_finish[,6]<-rep(1,N)*t_process
  weave_machine_finish[,7]<-rep(k,N)
  }
  else{
    t_process<-min_special(weave_machine[,2])
    k<-match(t_process,weave_machine[,2])
    weave_machine_finish<-weave_machine
    weave_machine_finish[,6]<-rep(1,N)*t_process
    weave_machine_finish[,7]<-rep(k,N)
  }
  return(weave_machine_finish)
}

#### function for finishing process ####
# N: number of finishing machines
# Q: queue length
# processing time ~ exp(120)
# status: 0 for idle, 1 for busy
finish<-function(finish_machine){
```

```
N<-length(finish_machine[,1])
avail<-N-sum(finish_machine[,1])
Q<-finish_machine[1,5]

if(Q>0 & avail>0){
j=1
if(avail<=Q){
  for(i in 1:N){
    if(finish_machine[i,1]==0 & j<=avail){
      finish_machine[i,1]=1
      finish_machine[i,2]<-rexp(1,rate=1/120)
      j=j+1
    }
  }
  finish_machine[,5]<-finish_machine[,5]-avail
}
else{
j=1
  for(i in 1:N){
    if(finish_machine[i,1]==0 & j<=Q){
      finish_machine[i,1]=1
      finish_machine[i,2]<-rexp(1,rate=1/120)
      j=j+1
    }
  }
  finish_machine[,5]<-finish_machine[,5]-Q
}
t_process<-min_special(finish_machine[,2])
k<-match(t_process,finish_machine[,2])
finish_machine_finish<-finish_machine
finish_machine_finish[,6]<-rep(1,N)*t_process
finish_machine_finish[,7]<-rep(k,N)
}
else{
  t_process<-min_special(finish_machine[,2])
  k<-match(t_process,finish_machine[,2])
  finish_machine_finish<-finish_machine
  finish_machine_finish[,6]<-rep(1,N)*t_process
  finish_machine_finish[,7]<-rep(k,N)
}

```

```
  return(finish_machine_finish)
}

#### function for packing process ####
# N: number of packing machines
# Q: queue length
# processing time ~ exp(360)
# status: 0 for idle, 1 for busy
pack<-function(pack_machine){
  N<-length(pack_machine[,1])
  avail<-N-sum(pack_machine[,1])
  Q<-pack_machine[1,5]

  if(Q>0 & avail>0){
  j=1
  if(avail<=Q){
    for(i in 1:N){
      if(pack_machine[i,1]==0 & j<=avail){
        pack_machine[i,1]=1
        pack_machine[i,2]<-rexp(1,rate=1/360)
        j=j+1
      }
    }
    pack_machine[,5]<-pack_machine[,5]-avail
  }
  else{
  j=1
    for(i in 1:N){
      if(pack_machine[i,1]==0 & j<=Q){
        pack_machine[i,1]=1
        pack_machine[i,2]<-rexp(1,rate=1/360)
        j=j+1
      }
    }
    pack_machine[,5]<-pack_machine[,5]-Q
  }
    t_process<-min_special(pack_machine[,2])
    k<-match(t_process,pack_machine[,2])
    pack_machine_finish<-pack_machine
    pack_machine_finish[,6]<-rep(1,N)*t_process
    pack_machine_finish[,7]<-rep(k,N)
```

```
  }
  else{
    t_process<-min_special(pack_machine[,2])
    k<-match(t_process,pack_machine[,2])
    pack_machine_finish<-pack_machine
    pack_machine_finish[,6]<-rep(1,N)*t_process
    pack_machine_finish[,7]<-rep(k,N)
  }
  return(pack_machine_finish)
}

#### function to handle connection between different stages ####
pass<-function(spin_machine,weave_machine,finish_machine,pack_machine){
  # check all process first
  spin_machine_check<-spin(spin_machine)
  weave_machine_check<-weave(weave_machine)
  finish_machine_check<-finish(finish_machine)
  pack_machine_check<-pack(pack_machine)
  # find the earliest finish time
  process<-c(spin_machine_check[1,6],weave_machine_check[1,6],
     finish_machine_check[1,6],pack_machine_check[1,6])
  real_process_time<-min_special(process)
  #find the earliest finish process
  real_process<-match(real_process_time,process)
  if(real_process==1){
    # release one machine in spin process
    spin_machine_out<-spin_machine_check
    spin_machine_out[,2]<-spin_machine_check[,2]-real_process_time*
       spin_machine_check[,1]
    spin_machine_out[,3]<-spin_machine_check[,3]+real_process_time*
       spin_machine_check[,1]
    spin_machine_out[,4]<-spin_machine_check[,4]+real_process_time
    spin_machine_out[spin_machine_out[1,7],1]<-0
    # add 1 to queue of weave process
    weave_machine_out<-weave_machine_check
    weave_machine_out[,2]<-weave_machine_check[,2]-real_process_time*weave_machine_check[,1]
    weave_machine_out[,3]<-weave_machine_check[,3]+real_process_time*weave_machine_check[,1]
```

```
    weave_machine_out[,4]<-weave_machine_check[,4]+real_process_time
    weave_machine_out[,5]<-weave_machine_check[,5]+1
    # let the time fly for finish process
    finish_machine_out<-finish_machine_check
    finish_machine_out[,2]<-finish_machine_check[,2]-real_process_time*finish_machine_check[,1]
    finish_machine_out[,3]<-finish_machine_check[,3]+real_process_time*finish_machine_check[,1]
    finish_machine_out[,4]<-finish_machine_check[,4]+real_process_time
    # let time fly for pack process
    pack_machine_out<-pack_machine_check
    pack_machine_out[,2]<-pack_machine_check[,2]-real_process_time*pack_machine_check[,1]
    pack_machine_out[,3]<-pack_machine_check[,3]+real_process_time*pack_machine_check[,1]
    pack_machine_out[,4]<-pack_machine_check[,4]+real_process_time
  }
  if(real_process==2){
    # let time fly for spin process
    spin_machine_out<-spin_machine_check
    spin_machine_out[,2]<-spin_machine_check[,2]-real_process_time*spin_machine_check[,1]
    spin_machine_out[,3]<-spin_machine_check[,3]+real_process_time*spin_machine_check[,1]
    spin_machine_out[,4]<-spin_machine_check[,4]+real_process_time
    # release one machine in weave process
    weave_machine_out<-weave_machine_check
    weave_machine_out[,2]<-weave_machine_check[,2]-real_process_time*weave_machine_check[,1]
    weave_machine_out[,3]<-weave_machine_check[,3]+real_process_time*weave_machine_check[,1]
    weave_machine_out[,4]<-weave_machine_check[,4]+real_process_time
    weave_machine_out[weave_machine_out[1,7],1]<-0
    # add 1 to quene of finish process
    finish_machine_out<-finish_machine_check
    finish_machine_out[,2]<-finish_machine_check[,2]-real_process_time*finish_machine_check[,1]
```

```
    finish_machine_out[,3]<-finish_machine_check[,3]+real_process_time*finish_machine_check[,1]
    finish_machine_out[,4]<-finish_machine_check[,4]+real_process_time
    finish_machine_out[,5]<-finish_machine_check[,5]+1
    # let time fly for pack process
    pack_machine_out<-pack_machine_check
    pack_machine_out[,2]<-pack_machine_check[,2]-real_process_time*pack_machine_check[,1]
    pack_machine_out[,3]<-pack_machine_check[,3]+real_process_time*pack_machine_check[,1]
    pack_machine_out[,4]<-pack_machine_check[,4]+real_process_time
  }
  if(real_process==3){
    # let time fly for spin process
    spin_machine_out<-spin_machine_check
    spin_machine_out[,2]<-spin_machine_check[,2]-real_process_time*spin_machine_check[,1]
    spin_machine_out[,3]<-spin_machine_check[,3]+real_process_time*spin_machine_check[,1]
    spin_machine_out[,4]<-spin_machine_check[,4]+real_process_time
    # let time fly for weave process
    weave_machine_out<-weave_machine_check
    weave_machine_out[,2]<-weave_machine_check[,2]-real_process_time*weave_machine_check[,1]
    weave_machine_out[,3]<-weave_machine_check[,3]+real_process_time*weave_machine_check[,1]
    weave_machine_out[,4]<-weave_machine_check[,4]+real_process_time
    # release 1 machine for finish process
    finish_machine_out<-finish_machine_check
    finish_machine_out[,2]<-finish_machine_check[,2]-real_process_time*finish_machine_check[,1]
    finish_machine_out[,3]<-finish_machine_check[,3]+real_process_time*finish_machine_check[,1]
    finish_machine_out[,4]<-finish_machine_check[,4]+real_process_time
    finish_machine_out[finish_machine_out[1,7],1]<-0
    # add 1 to queue of pack process
    pack_machine_out<-pack_machine_check
```

```
    pack_machine_out[,2]<-pack_machine_check[,2]-real_process_time*pack_machine_check[,1]
    pack_machine_out[,3]<-pack_machine_check[,3]+real_process_time*pack_machine_check[,1]
    pack_machine_out[,4]<-pack_machine_check[,4]+real_process_time
    pack_machine_out[,5]<-pack_machine_check[,5]+1
  }
  if(real_process==4){
    # add 1 to the queue of spin process
    spin_machine_out<-spin_machine_check
    spin_machine_out[,2]<-spin_machine_check[,2]-real_process_time*spin_machine_check[,1]
    spin_machine_out[,3]<-spin_machine_check[,3]+real_process_time*spin_machine_check[,1]
    spin_machine_out[,4]<-spin_machine_check[,4]+real_process_time
    spin_machine_out[,5]<-spin_machine_check[,5]+1
    # let time fly for weave process
    weave_machine_out<-weave_machine_check
    weave_machine_out[,2]<-weave_machine_check[,2]-real_process_time*weave_machine_check[,1]
    weave_machine_out[,3]<-weave_machine_check[,3]+real_process_time*weave_machine_check[,1]
    weave_machine_out[,4]<-weave_machine_check[,4]+real_process_time
    # let time fly for finish process
    finish_machine_out<-finish_machine_check
    finish_machine_out[,2]<-finish_machine_check[,2]-real_process_time*finish_machine_check[,1]
    finish_machine_out[,3]<-finish_machine_check[,3]+real_process_time*finish_machine_check[,1]
    finish_machine_out[,4]<-finish_machine_check[,4]+real_process_time
    # release 1 machine for pack process
    pack_machine_out<-pack_machine_check
    pack_machine_out[,2]<-pack_machine_check[,2]-real_process_time*pack_machine_check[,1]
    pack_machine_out[,3]<-pack_machine_check[,3]+real_process_time*pack_machine_check[,1]
    pack_machine_out[,4]<-pack_machine_check[,4]+real_process_time
    pack_machine_out[pack_machine_out[1,7],1]<-0
  }
```

```
  result<-list(spin_machine_out,weave_machine_out,finish_machine_out,pack_machine_out)
  return(result)
}

#### Main simulation function ####
simulation<-function(N_A,N_B,N_C,N_D,Q_0=25){
  spin_machine_begin<-matrix(0,N_A,7)
  weave_machine_begin<-matrix(0,N_B,7)
  finish_machine_begin<-matrix(0,N_C,7)
  pack_machine_begin<-matrix(0,N_D,7)
  spin_machine_begin[,5]<-Q_0*rep(1,N_A)
  while (t_clock<=t_end) {
    status<-pass(spin_machine_begin,weave_machine_begin,finish_machine_begin,pack_machine_begin)
    spin_machine_begin<-status[[1]]
    weave_machine_begin<-status[[2]]
    finish_machine_begin<-status[[3]]
    pack_machine_begin<-status[[4]]
    t_clock<-status[[1]][1,4]
  }
  occupy_rate<-sum(sum(status[[1]][,3]),sum(status[[2]][,3]),sum(status[[3]][,3]),sum(status[[4]][,3]))/sum(sum(status[[1]][,4]),sum(status[[2]][,4]),sum(status[[3]][,4]),sum(status[[4]][,4]))
  idle_rate<-1-occupy_rate
  result<-c(N_A,N_B,N_C,N_D,idle_rate)
  return(result)
}


###### Part 2: Simulation start ######

#### Initialization ####
t_clock<-0 #clock time
t_end=7*24*3600 #end time
Q_0=25 #initial queue length
result_table<-data.frame("Spinning"=0,"Weaving"=0,"Finishing"=0,"Packing"=0,"Avg_idle_rate"=0) #table to store results

#### Loop all combinations to find optimal solution ####
```

```r
for (a in 1:10){
  for (b in 1:10){
    for (c in 1:10){
      for (d in 1:10){
        result_table<-rbind(result_table,simulation(a,b,c,d))
      }
    }
  }
}
result_table<-result_table[-c(1),]

#### Sort result table ####

#### Export result table to csv file ####
write.csv(result_table,file="results.csv",row.names=FALSE)
```