User Guide for the Simulation-based Virtual Testbed for Smart Manufacturing

1. Introduction

Background: The simulation-based virtual testbed for smart manufacturing was develop as part of the research project sponsored by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office Award Number DE-EE0007613: *Energy Management Systems for Subtractive and Additive Precision Manufacturing*.

Purpose: The testbed is developed to integrate mathematical models and decision-making, optimization tools at different levels of a manufacturing process into one unified computational framework.

Team: The testbed was developed jointly by

- Ms. Yuting Sun, PhD student (expected May 2024), University of Connecticut
- Mr. Jiachen Tu, PhD student (expected May 2023), University of Connecticut
- Dr. Mikhail Bragin, Assistant Research Professor, University of Connecticut
- Dr. Liang Zhang[1], Associate Professor, University of Connecticut

2. Computing Environment and Requirement

The testbed was developed to run on a Windows PC. It requires the following software:

- Python 3.7: An interpreted high-level general-purpose programming language.
- Simul8: A commercial discrete-event simulation software package developed by Simul8 Corporation (https://www.simul8.com/).
- IBM ILOG CPLEX Optimization Studio (referred to as CPLEX): An optimization solver software package developed by IBM (https://www.ibm.com/products/ilog-cplex-optimization-studio).
- MS Excel: Spreadsheet tool in MS Office suite.

To execute the program in Python, Visual Studio Code (https://code.visualstudio.com/) environment is recommended and used by the developer. In addition, the following packages need to be installed in Python in order to run the testbed program:

- pywin32: This library provides access to many of the Windows APIs from Python. In this application, it is used to access Simul8 and control the start of simulation.
- doopl: It is a package to run OPL (Optimization Programming Language) models from python.
- os: os module is included in the standard library, so no additional installation is required. You can get and change (set) the current working directory with os.getcwd() and os.chdir().
- math: This module provides access to the mathematical functions defined by the C standard.
- threading: This module provides threading functions to run multiple I/O-bound tasks simultaneously.
- time: This module provides various time-related functions. In this application, time.sleep() is used.

[1] Please contact Dr. Liang Zhang (liang.zhang@uconn.edu) for any question/inquiry about the testbed.

- numpy: It is an open source package aiming to enable numerical computing with Python.
- openpyxl: It is a Python library to read/write Excel files.
- pandas: It is a Python library used to deal with data.

3. Manufacturing System Models

The source code and programs submitted are for the manufacturing described below. As illustrated in Figure 1, the system consists of 5 operations: Op. 1 Lathing, Op. 2 Milling, Op. 3 Drilling, Op. 4 Grinding, Op. 5 Assembly. Each machine has certain number of parallel machines (indicated as circles in the figure). Consecutive operations are separated by buffers of sufficiently large capacity. The system is to mimic a typical machine shop in discrete part manufacturing industries.

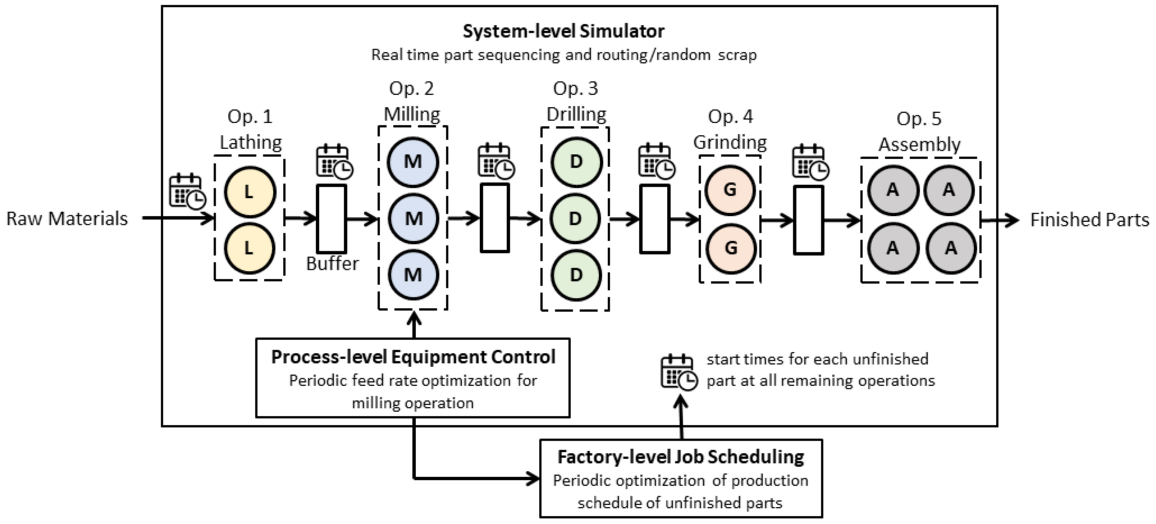

System-level Simulator
Real time part sequencing and routing/random scrap
Raw Materials
Op. 1
Lathing
L
L
Buffer
Op. 2
Milling
M
M
M
Op. 3
Drilling
D
D
D
Op. 4
Grinding
G
G
Op. 5
Assembly
A
A
A
A
Finished Parts
Process-level Equipment Control
Periodic feed rate optimization for milling operation
start times for each unfinished part at all remaining operations
Factory-level Job Scheduling
Periodic optimization of production schedule of unfinished parts

Figure 1. Manufacturing system model flow diagram

Briefly describe how each level operates,

- For the process-level model, give all the formulas, with very clear but brief explanation (refer to 3.2 in the paper) In this case study, we assume that the feed rate of the *Milling* operation is adjustable, while other parameters (depth of cut, width of cut, etc.) are fixed. The operator can change the feed rate to optimize the performance of this operation and reduce the processing cost. This is conducted periodically to account for the workload and tool wear of the machines. For other operations, we assume that their parameters are fixed.
- For the scheduling model, focus on the input/output parameters (refer to 3.3 in the paper), no need to give the constraints. Like a typical machine shop, this system is tasked to produce a certain number of parts of different product types. The scheduler's role is to determine when each part should be processed at each operation to achieve certain objectives (overall completion time, on-time delivery, etc.). Note that this is also carried out periodically based on the production status, parts scrapped, and adjusted processing times of the machines on the process level.
- For the system-level model, focus on how it operates and how it interacts with the other two models (refer to first four paragraphs of 3.5 in the paper, SCM will be discussed in the next

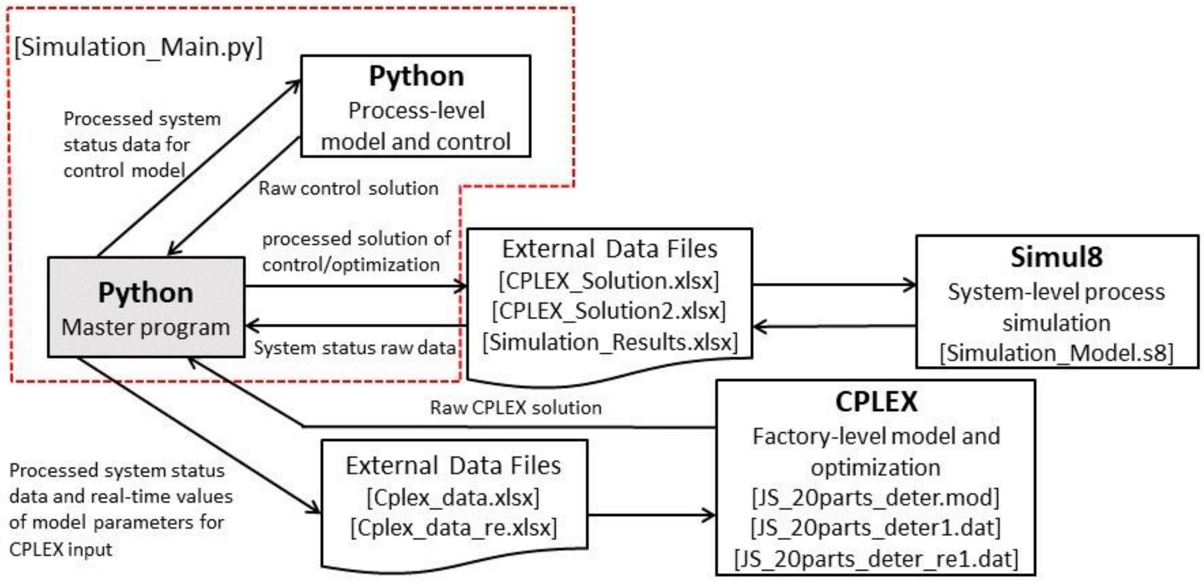
section of this document). At the system level, the control amounts to the routing and prioritizing the parts arriving at an operation to specific machines in real time to counter any unexpected events (e.g, part scraps). It needs to take into account the scheduled start times of the parts from the factory level and the actual processing outcomes to determine which part is processed next on each operation and when.

4. Computational Implementation

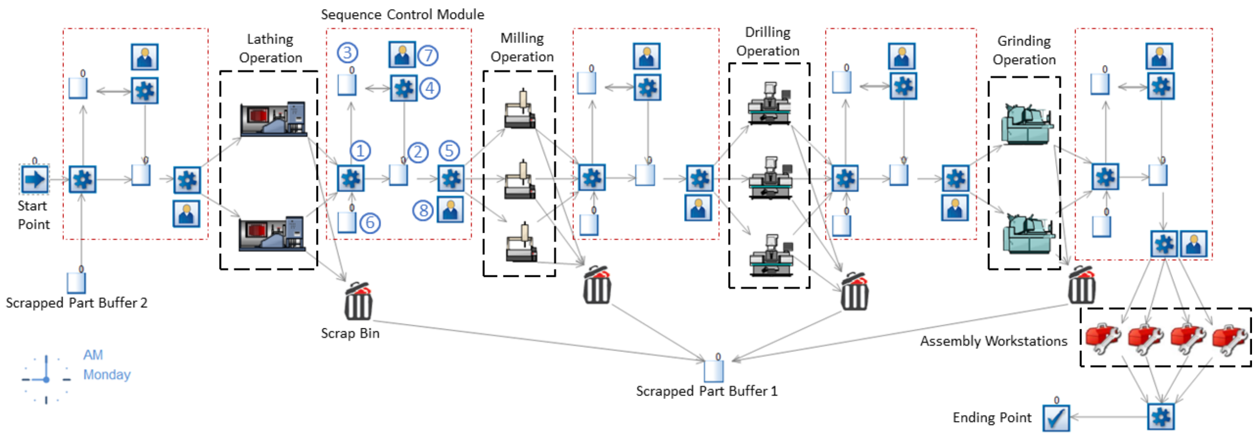
4.1 List of files

List files:

File name	Platform	Notes
Simulation_Main.py	Python	Master program:
Simulation_Model.s8	Simul8	Simulation
JS_20parts_deter.mod	CPLEX	CPLEX model file
JS_20parts_deter1.dat	CPLEX	CPLEX data file for initial scheduling
JS_20parts_deter_re1.dat	CPLEX	CPLEX data file for rescheduling
Cplex_data.xlsx	MS Excel	Supplementary data file for initial scheduling
Cplex_data_re.xlsx	MS Excel	Supplementary data file for rescheduling
CPLEX_Solution.xlsx	MS Excel	Excel file for data interaction between Simul8 and Python
Simulation_Results.xlsx	MS Excel	Excel file for results recording of Simul8
CPLEX_Solution2.xlsx	MS Excel	The copy of CPLEX_Solution.xlsx

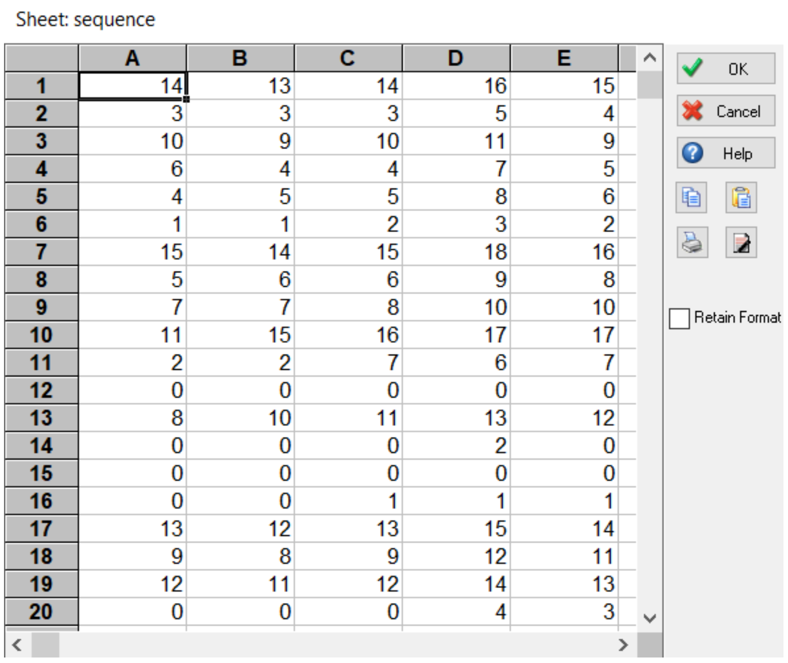

[Simulation_Main.py]
Python
Process-level model and control
Processed system status data for control model
Raw control solution
processed solution of control/optimization
Python
Master program
System status raw data
External Data Files
[CPLEX_Solution.xlsx]
[CPLEX_Solution2.xlsx]
[Simulation_Results.xlsx]
Simul8
System-level process simulation
[Simulation_Model.s8]
Raw CPLEX solution
CPLEX
Factory-level model and optimization
[JS_20parts_deter.mod]
[JS_20parts_deter1.dat]
[JS_20parts_deter_re1.dat]
Processed system status data and real-time values of model parameters for CPLEX input
External Data Files
[Cplex_data.xlsx]
[Cplex_data_re.xlsx]

4.2 Simulation module



In the Simul8 model, the parts to be processed are generated by the Start Point. The five operations in the system -- Lathing, Milling, Drilling, Grinding, and Assembly -- are represented by different icons. The scrapped parts from each operation are collected in its Scrap Bin and then to Scrapped Parts Buffer 1 during shifts and are transferred to Scrapped Parts Buffer 2 at the end of each shift to report the parts to be restarted during rescheduling by the factory-level scheduler.

After the Simul8 model receives the scheduled beginning times, it converts them into part priority sequences (PPS) for each operation (see figure ...). For the same operation, earlier beginning time earns higher priority for the part in the sequence and lower priority parts must wait for the highest priority parts to enter the operation first. Note that if a part is scrapped in an operation, it is deleted from the part priority sequences in all subsequent operations.



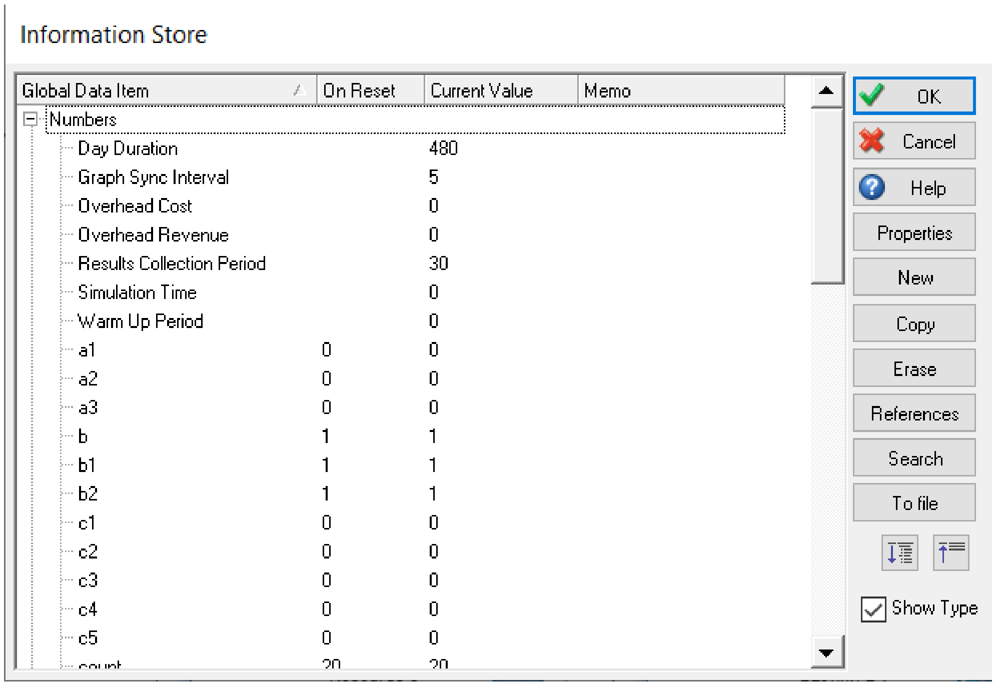
Sheet: sequence

	A	B	C	D	E
1	14	13	14	16	15
2	3	3	3	5	4
3	10	9	10	11	9
4	6	4	4	7	5
5	4	5	5	8	6
6	1	1	2	3	2
7	15	14	15	18	16
8	5	6	6	9	8
9	7	7	8	10	10
10	11	15	16	17	17
11	2	2	7	6	7
12	0	0	0	0	0
13	8	10	11	13	12
14	0	0	0	2	0
15	0	0	0	0	0
16	0	0	1	1	1
17	13	12	13	15	14
18	9	8	9	12	11
19	12	11	12	14	13
20	0	0	0	4	3

Sequence Control Module: Between two consecutive operations, a Sequence Control Module (SCM) is created (see a red box with a dot-dash line). The function of SCM is to sort the parts finished by the upstream operation based on the PPS of the downstream operation and control the release of the parts to the downstream machines. In the Simul8 model, an SCM is realized using a number of virtual units: three virtual machines with 0 processing time (units 1, 4 and 5), three virtual buffers (units 2, 3 and 6), and two virtual resources (units 7 and 8) accompanied by several customized code modules programmed in Visual Logic, the built-in programming language of Simul8. Specifically, virtual machine 1 routes the parts with the highest priority in the PPS to virtual buffer 2. If a part with lower priority arrives at machine 1 first, it is temporarily routed to buffer 3. Every time a high priority part enters buffer 2, resource 7 authorizes virtual machine 4 to scan the parts in buffer 3, determines if the next part in the PPS is in the buffer, moves it to buffer 2, and continues until no such movement is possible. Resource 8 authorizes virtual machine 5 to release a part from buffer 2 to the next operation once a machine becomes available to receive. Virtual machine 5 also records the departure information (timing and destination) of the parts as they exit the SCM. At the end of each shift, all remaining parts in buffers 2 and 3 are moved to buffer 6. Once the new PPS is generated based on the reschedule, the parts will be again examined by machines 1 and 4 and routed to buffers 2 and 3 accordingly.

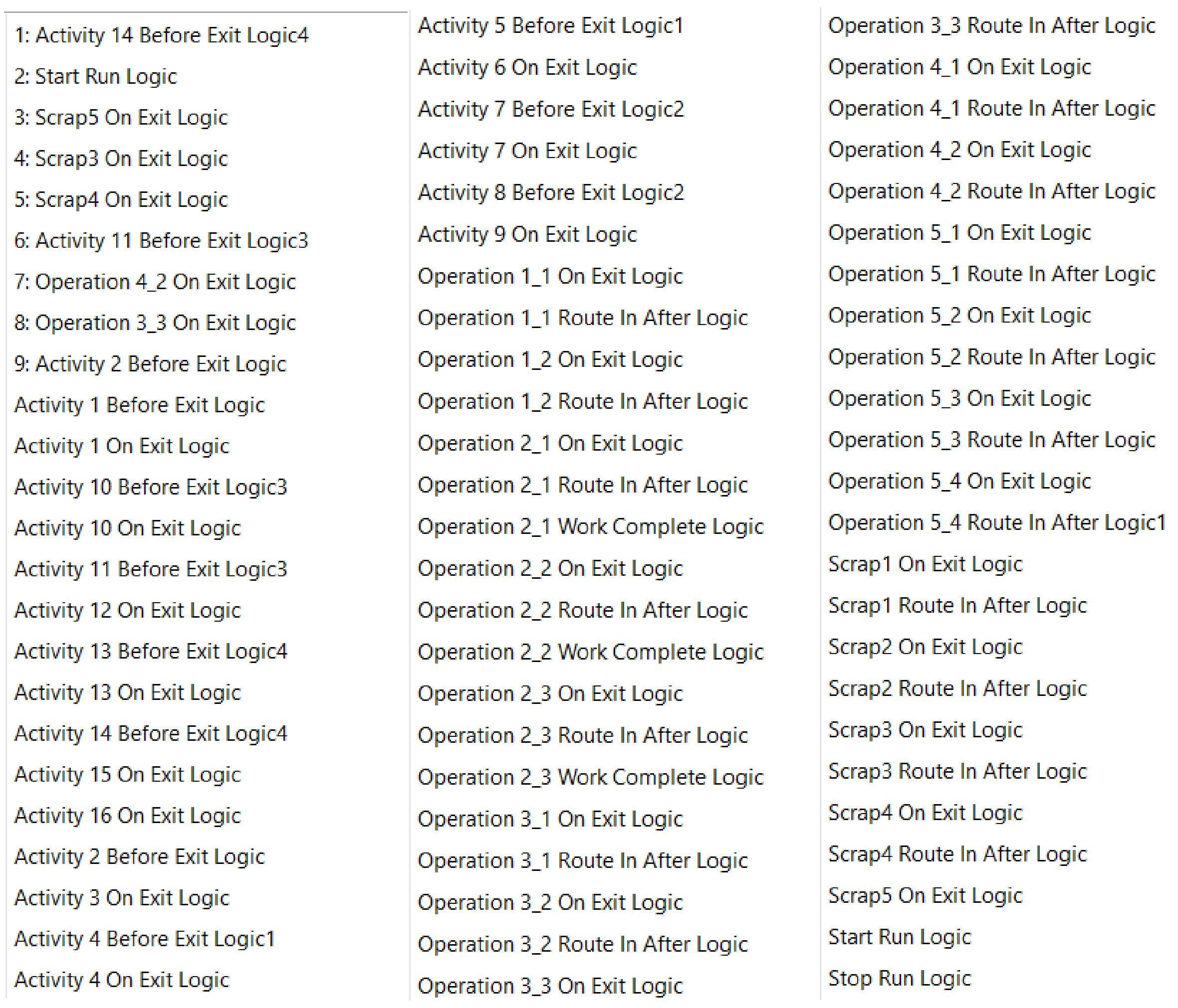
The implementation of this part is based on visual logic code in Simul8. The following part will illustrate the details of the code.

The variables used in code are defined in Information Store.



Global Data Item	On Reset	Current Value	Memo
Numbers			
Day Duration		480	
Graph Sync Interval		5	
Overhead Cost		0	
Overhead Revenue		0	
Results Collection Period		30	
Simulation Time		0	
Warm Up Period		0	
a1	0	0	
a2	0	0	
a3	0	0	
b	1	1	
b1	1	1	
b2	1	1	
c1	0	0	
c2	0	0	
c3	0	0	
c4	0	0	
c5	0	0	
count	20	20	

Here is the index of the visual logic code.



1: Activity 14 Before Exit Logic4
2: Start Run Logic
3: Scrap5 On Exit Logic
4: Scrap3 On Exit Logic
5: Scrap4 On Exit Logic
6: Activity 11 Before Exit Logic3
7: Operation 4_2 On Exit Logic
8: Operation 3_3 On Exit Logic
9: Activity 2 Before Exit Logic
Activity 1 Before Exit Logic
Activity 1 On Exit Logic
Activity 10 Before Exit Logic3
Activity 10 On Exit Logic
Activity 11 Before Exit Logic3
Activity 12 On Exit Logic
Activity 13 Before Exit Logic4
Activity 13 On Exit Logic
Activity 14 Before Exit Logic4
Activity 15 On Exit Logic
Activity 16 On Exit Logic
Activity 2 Before Exit Logic
Activity 3 On Exit Logic
Activity 4 Before Exit Logic1
Activity 4 On Exit Logic
Activity 5 Before Exit Logic1
Activity 6 On Exit Logic
Activity 7 Before Exit Logic2
Activity 7 On Exit Logic
Activity 8 Before Exit Logic2
Activity 9 On Exit Logic
Operation 1_1 On Exit Logic
Operation 1_1 Route In After Logic
Operation 1_2 On Exit Logic
Operation 1_2 Route In After Logic
Operation 2_1 On Exit Logic
Operation 2_1 Route In After Logic
Operation 2_1 Work Complete Logic
Operation 2_2 On Exit Logic
Operation 2_2 Route In After Logic
Operation 2_2 Work Complete Logic
Operation 2_3 On Exit Logic
Operation 2_3 Route In After Logic
Operation 2_3 Work Complete Logic
Operation 3_1 On Exit Logic
Operation 3_1 Route In After Logic
Operation 3_2 On Exit Logic
Operation 3_2 Route In After Logic
Operation 3_3 On Exit Logic
Operation 3_3 Route In After Logic
Operation 4_1 On Exit Logic
Operation 4_1 Route In After Logic
Operation 4_2 On Exit Logic
Operation 4_2 Route In After Logic
Operation 5_1 On Exit Logic
Operation 5_1 Route In After Logic
Operation 5_2 On Exit Logic
Operation 5_2 Route In After Logic
Operation 5_3 On Exit Logic
Operation 5_3 Route In After Logic
Operation 5_4 On Exit Logic
Operation 5_4 Route In After Logic1
Scrap1 On Exit Logic
Scrap1 Route In After Logic
Scrap2 On Exit Logic
Scrap2 Route In After Logic
Scrap3 On Exit Logic
Scrap3 Route In After Logic
Scrap4 On Exit Logic
Scrap4 Route In After Logic
Scrap5 On Exit Logic
Start Run Logic
Stop Run Logic

Start Run Logic: The code initializes the programming variables in Simul8. Then, read processing time, sequence, and scrap rate from excel sheets.

Activity 1/4/7/10/13 Before Exit Logic: The code determines if the current part is the next processing part based on the sequence sheet, if it is the next part, the label 2 of the part will be set to 1, otherwise label 2 will be set to 2.

Activity 2/5/8/11/14 On Exit Logic: When a part has been released by the corresponding activity 1/4/7/10/13, scan the parts in the previous buffer, determines if the next part in the PPS is in the buffer.

Activity 3/6/9/12/15 On Exit Logic: Record the processing start time to excel file for the next operation.

Activity 4/7/10/13 On Exit Logic: Record the processing end time to excel file for the previous operation.

Activity 1 On Exit Logic: Count the number of parts which have been released.

Operation x_x On Exit Logic: Count the number of parts which have been finished from the current operation.

Operation x_x Route In After Logic: 1. Read the processing time of current loaded part. 2. If the occupation of the current operation is full. Lock activity 3/6/9/12/15 through resource 6/7/8/9/10 so that the corresponding activity will not load the part. This is helpful because the parts in activity can not be moved when a time shift finished.

Scrap1/2/3/4/5 Route In After Logic: 1. If a part is scrapped, the priority sequence in the sequence sheet will be set to 0. 2. Count the number of parts scrapped in the current operation.

Scrap1/2/3/4/5 On Exit Logic: Record the processing end time to excel file for the previous operation.

Stop Run Logic: Move all scraped parts from scrap queue to the corresponding buffer.

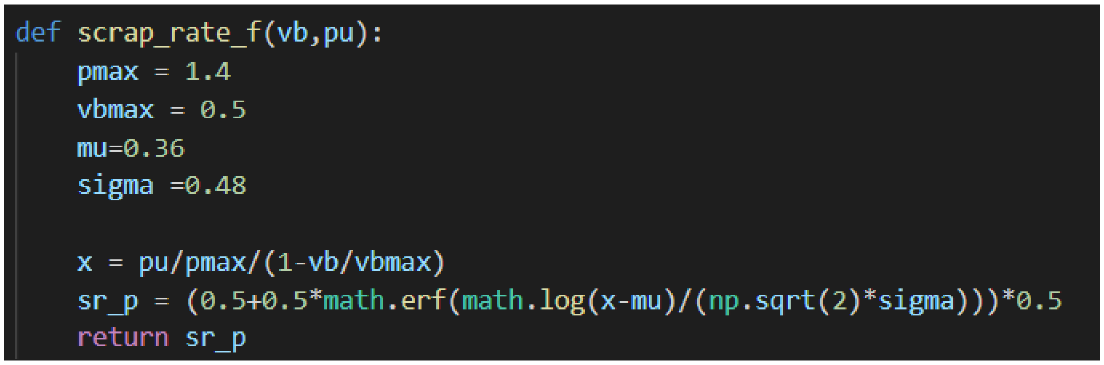
4.3 Master program

Function and implementation: Master program is a main program integrating the simulation of Simul8, optimization of process level model and scheduling optimization by CPLEX module. It is a simulation-based integrated virtual testbed for dynamic optimization of manufacturing systems with parameter coupling, objective alignment, and automated and synchronized data exchange. The process-level model calculates the optimized feed rate of the Milling operation and send the corresponding processing time to the system-level simulator and the factory-level scheduler. CPLEX module computes the initial optimized schedule of all parts at all operations. With the scrape rate of milling machines and the part priority sequences of the operations determined, simulation is started in the testbed. After the shift, the system status at the end of the shift will be exported from the simulator and passed on to the process-level model and the factory-level scheduler. According to the previous simulation results, repeat previous procedures until all parts are finished.

#Part 1

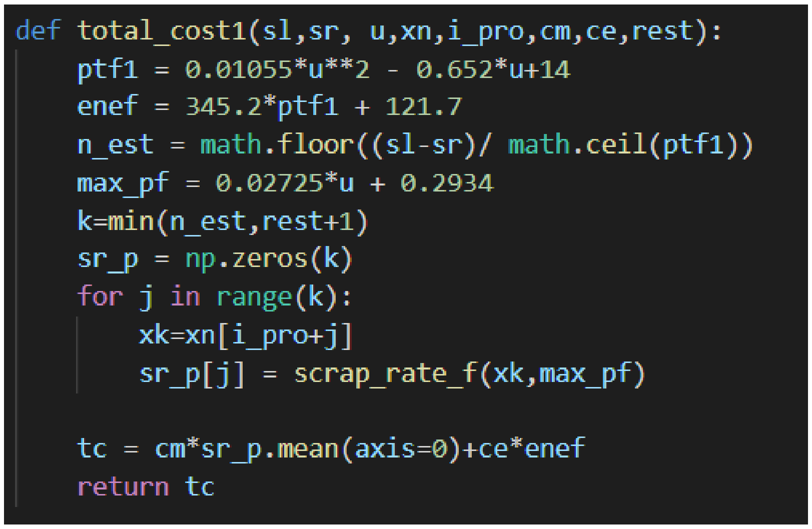
Functions in master program:

scrap_rate_f: calculate the scrap_rate of operation 2.



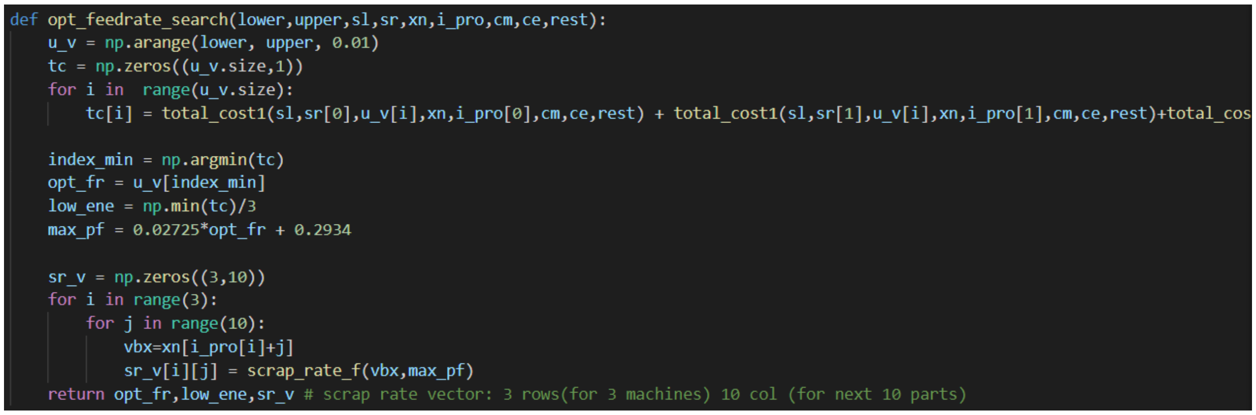
```
def scrap_rate_f(vb,pu):
    pmax = 1.4
    vbmax = 0.5
    mu=0.36
    sigma =0.48

    x = pu/pmax/(1-vb/vbmax)
    sr_p = (0.5+0.5*math.erf(math.log(x-mu)/(np.sqrt(2)*sigma)))*0.5
    return sr_p
```

total_cost1: calculate the cost of operation 2.



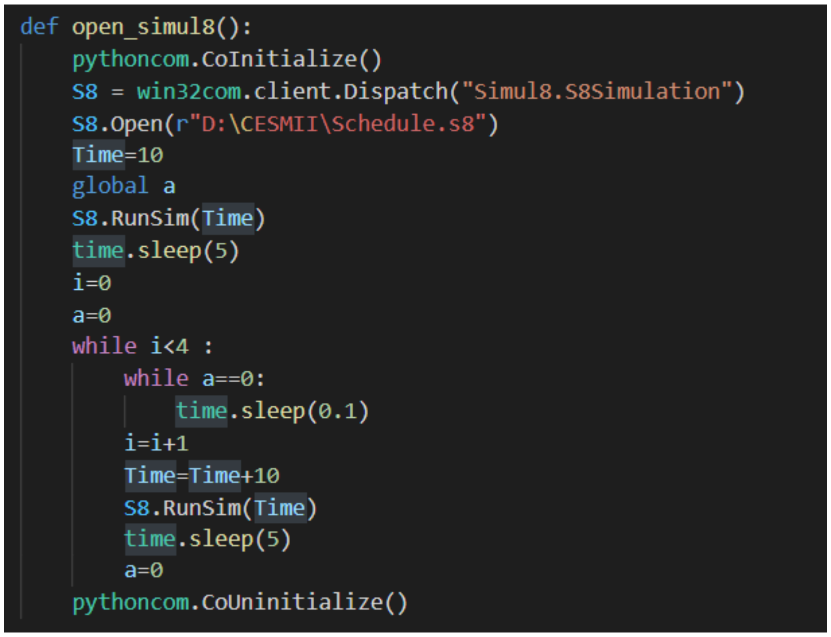
```python
def total_cost1(sl,sr, u,xn,i_pro,cm,ce,rest):
    ptf1 = 0.01055*u**2 - 0.652*u+14
    enef = 345.2*ptf1 + 121.7
    n_est = math.floor((sl-sr)/ math.ceil(ptf1))
    max_pf = 0.02725*u + 0.2934
    k=min(n_est,rest+1)
    sr_p = np.zeros(k)
    for j in range(k):
        xk=xn[i_pro+j]
        sr_p[j] = scrap_rate_f(xk,max_pf)

    tc = cm*sr_p.mean(axis=0)+ce*enef
    return tc
```

opt_feedrate_search: find the optimal feed rate.



```python
def opt_feedrate_search(lower,upper,sl,sr,xn,i_pro,cm,ce,rest):
    u_v = np.arange(lower, upper, 0.01)
    tc = np.zeros((u_v.size,1))
    for i in  range(u_v.size):
        tc[i] = total_cost1(sl,sr[0],u_v[i],xn,i_pro[0],cm,ce,rest) + total_cost1(sl,sr[1],u_v[i],xn,i_pro[1],cm,ce,rest)+total_cos

    index_min = np.argmin(tc)
    opt_fr = u_v[index_min]
    low_ene = np.min(tc)/3
    max_pf = 0.02725*opt_fr + 0.2934

    sr_v = np.zeros((3,10))
    for i in range(3):
        for j in range(10):
            vbx=xn[i_pro[i]+j]
            sr_v[i][j] = scrap_rate_f(vbx,max_pf)
    return opt_fr,low_ene,sr_v # scrap rate vector: 3 rows(for 3 machines) 10 col (for next 10 parts)
```

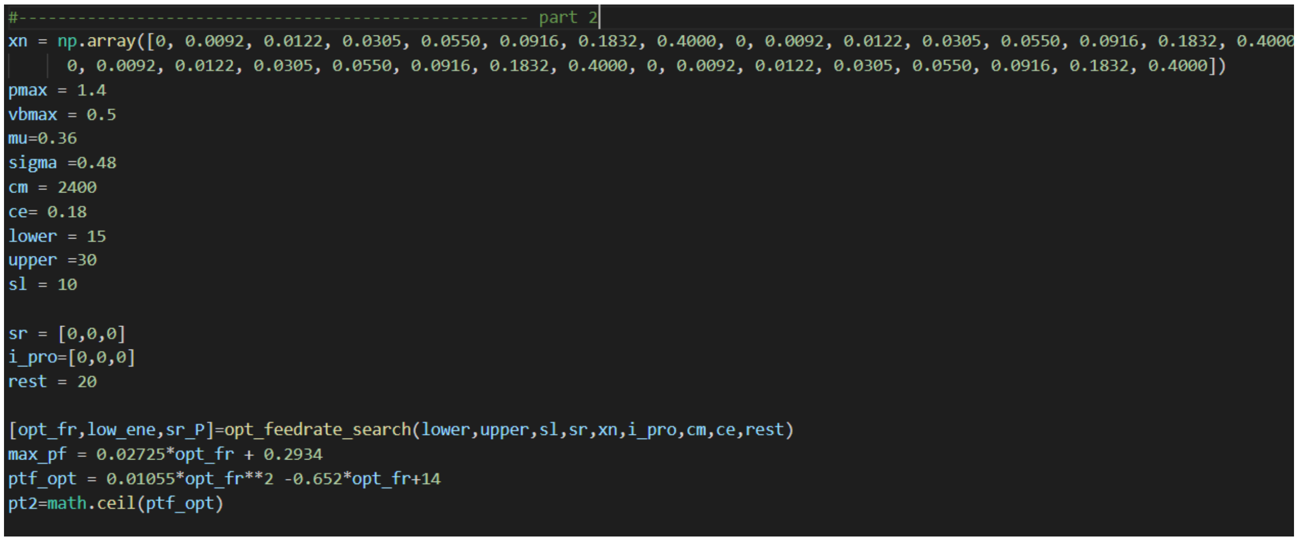
open_simul8(): the sub thread which is used to run Simul8.



```python
def open_simul8():
    pythoncom.CoInitialize()
    S8 = win32com.client.Dispatch("Simul8.S8Simulation")
    S8.Open(r"D:\CESMII\Schedule.s8")
    Time=10
    global a
    S8.RunSim(Time)
    time.sleep(5)
    i=0
    a=0
    while i<4 :
        while a==0:
            time.sleep(0.1)
        i=i+1
        Time=Time+10
        S8.RunSim(Time)
        time.sleep(5)
        a=0
    pythoncom.CoUninitialize()
```

#Part 2

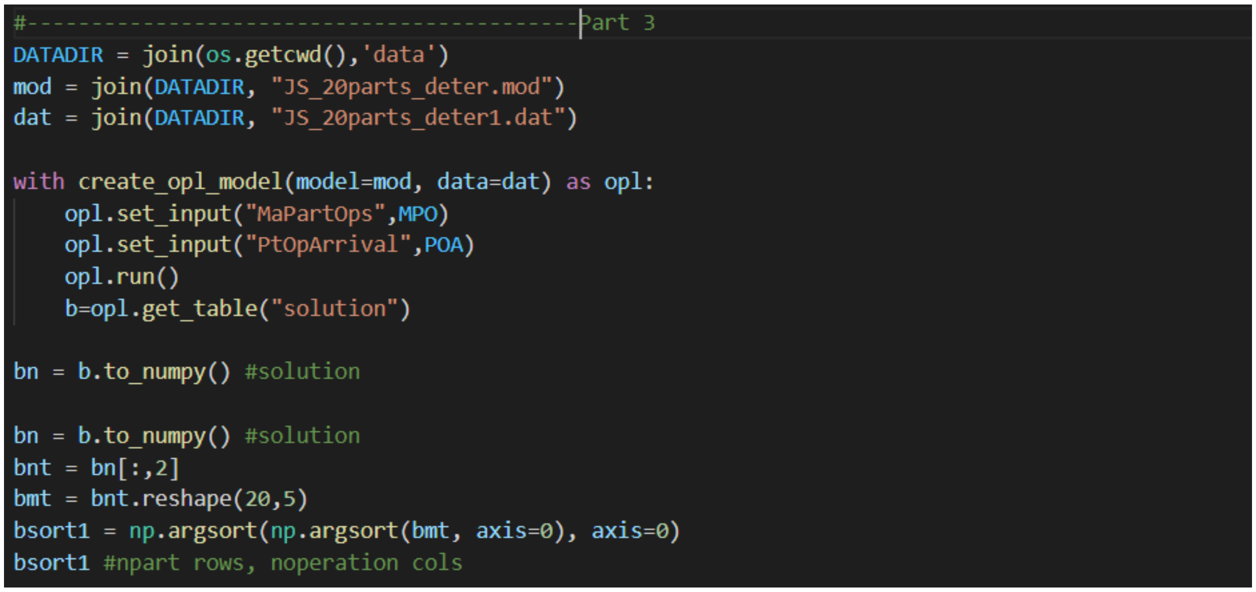
This part of code defines the parameters used in optimization of process level and calculate the solution.



```
#------------------------------------------------- part 2
xn = np.array([0, 0.0092, 0.0122, 0.0305, 0.0550, 0.0916, 0.1832, 0.4000, 0, 0.0092, 0.0122, 0.0305, 0.0550, 0.0916, 0.1832, 0.4000
      0, 0.0092, 0.0122, 0.0305, 0.0550, 0.0916, 0.1832, 0.4000, 0, 0.0092, 0.0122, 0.0305, 0.0550, 0.0916, 0.1832, 0.4000])
pmax = 1.4
vbmax = 0.5
mu=0.36
sigma =0.48
cm = 2400
ce= 0.18
lower = 15
upper =30
sl = 10

sr = [0,0,0]
i_pro=[0,0,0]
rest = 20

[opt_fr,low_ene,sr_P]=opt_feedrate_search(lower,upper,sl,sr,xn,i_pro,cm,ce,rest)
max_pf = 0.02725*opt_fr + 0.2934
ptf_opt = 0.01055*opt_fr**2 -0.652*opt_fr+14
pt2=math.ceil(ptf_opt)
```

#Part 3

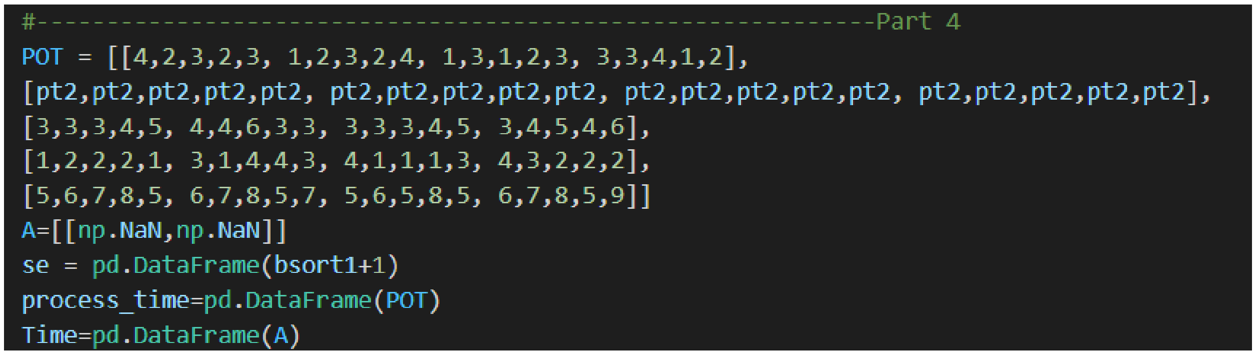
Part 3 realizes the running of CPLEX module and deals with the solution. Get processing time, sequence etc.



```
#------------------------------------------Part 3
DATADIR = join(os.getcwd(),'data')
mod = join(DATADIR, "JS_20parts_deter.mod")
dat = join(DATADIR, "JS_20parts_deter1.dat")

with create_opl_model(model=mod, data=dat) as opl:
    opl.set_input("MaPartOps",MPO)
    opl.set_input("PtOpArrival",POA)
    opl.run()
    b=opl.get_table("solution")

bn = b.to_numpy() #solution

bn = b.to_numpy() #solution
bnt = bn[:,2]
bmt = bnt.reshape(20,5)
bsort1 = np.argsort(np.argsort(bmt, axis=0), axis=0)
bsort1 #npart rows, noperation cols
```

#Part 4

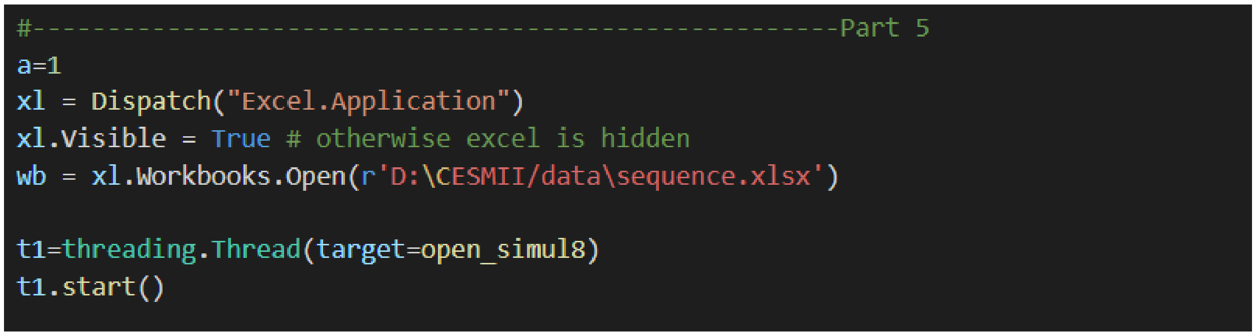
Part 4 converts the data into Excel file "CPLEX_Solution.xlsx".



```python
#----------------------------------------------------------Part 4
POT = [[4,2,3,2,3, 1,2,3,2,4, 1,3,1,2,3, 3,3,4,1,2],
[pt2,pt2,pt2,pt2,pt2, pt2,pt2,pt2,pt2,pt2, pt2,pt2,pt2,pt2,pt2, pt2,pt2,pt2,pt2,pt2],
[3,3,3,4,5, 4,4,6,3,3, 3,3,3,4,5, 3,4,5,4,6],
[1,2,2,2,1, 3,1,4,4,3, 4,1,1,1,3, 4,3,2,2,2],
[5,6,7,8,5, 6,7,8,5,7, 5,6,5,8,5, 6,7,8,5,9]]
A=[[np.NaN,np.NaN]]
se = pd.DataFrame(bsort1+1)
process_time=pd.DataFrame(POT)
Time=pd.DataFrame(A)
```

#Part 5

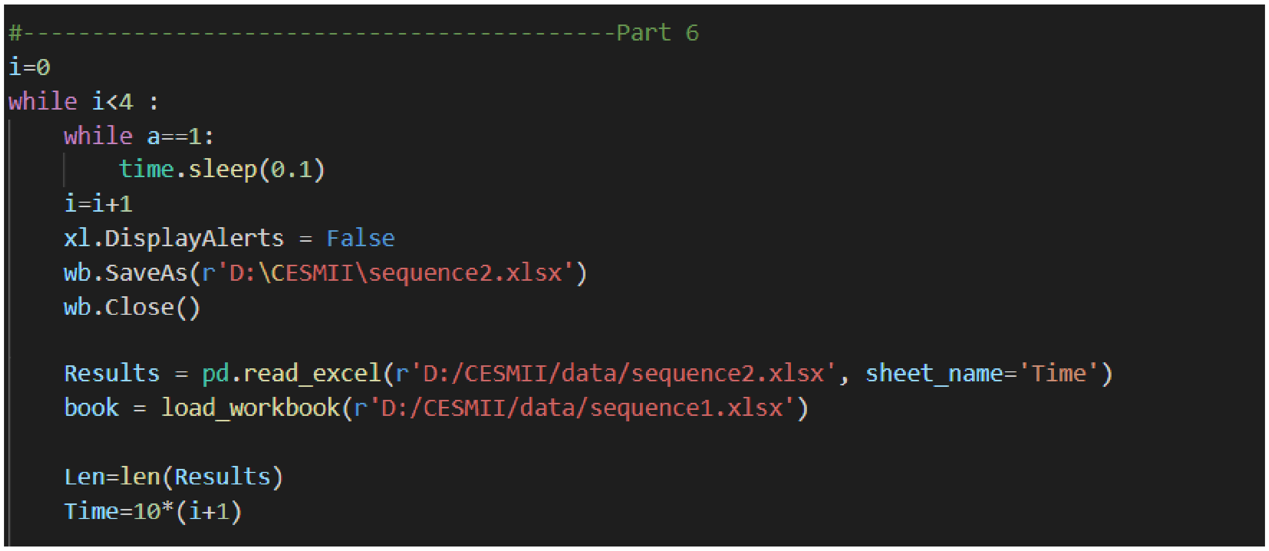
This part opens the excel application and build a new thread to run Simul8. Note the variable ‘a’ is a flag that indicates which parts should be running. If a=1, Simul8 will do simulation. If a=0, optimization of rescheduling will work.



```python
#----------------------------------------------------Part 5
a=1
xl = Dispatch("Excel.Application")
xl.Visible = True # otherwise excel is hidden
wb = xl.Workbooks.Open(r'D:\CESMII/data\sequence.xlsx')

t1=threading.Thread(target=open_simul8)
t1.start()
```

#Part 6

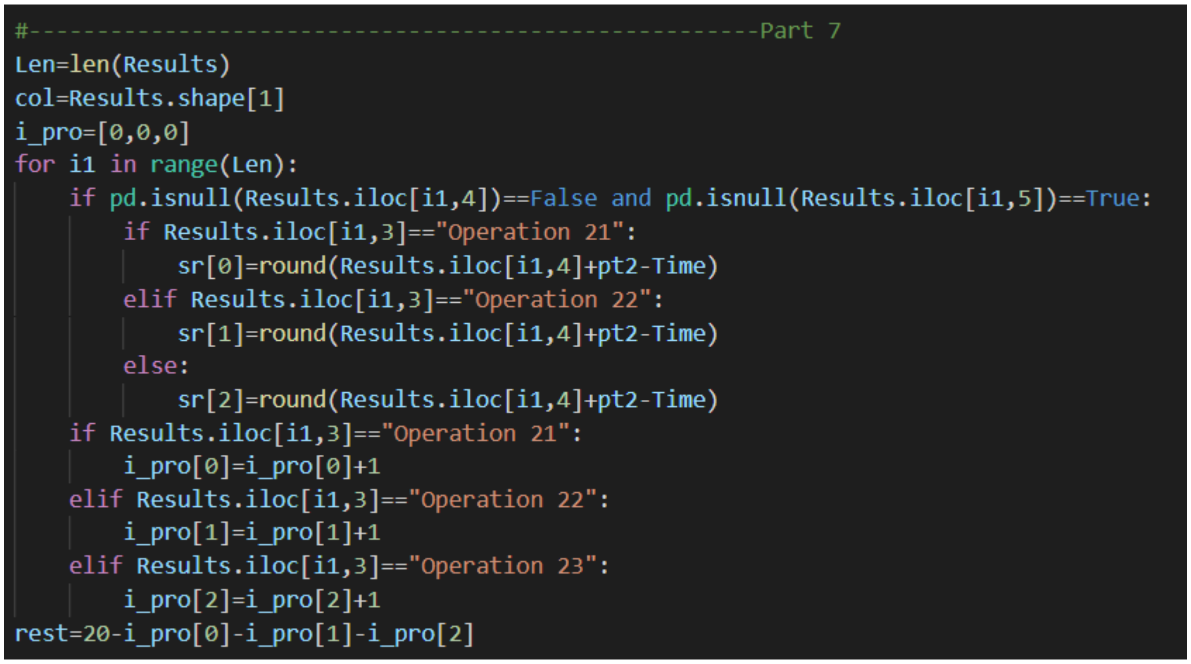
Wait the simulation. When simulation finished, read the results of simulation from the excel file.



```python
#-----------------------------------------Part 6
i=0
while i<4 :
    while a==1:
        time.sleep(0.1)
    i=i+1
    xl.DisplayAlerts = False
    wb.SaveAs(r'D:\CESMII\sequence2.xlsx')
    wb.Close()

    Results = pd.read_excel(r'D:/CESMII/data/sequence2.xlsx', sheet_name='Time')
    book = load_workbook(r'D:/CESMII/data/sequence1.xlsx')

    Len=len(Results)
    Time=10*(i+1)
```

#Part 7

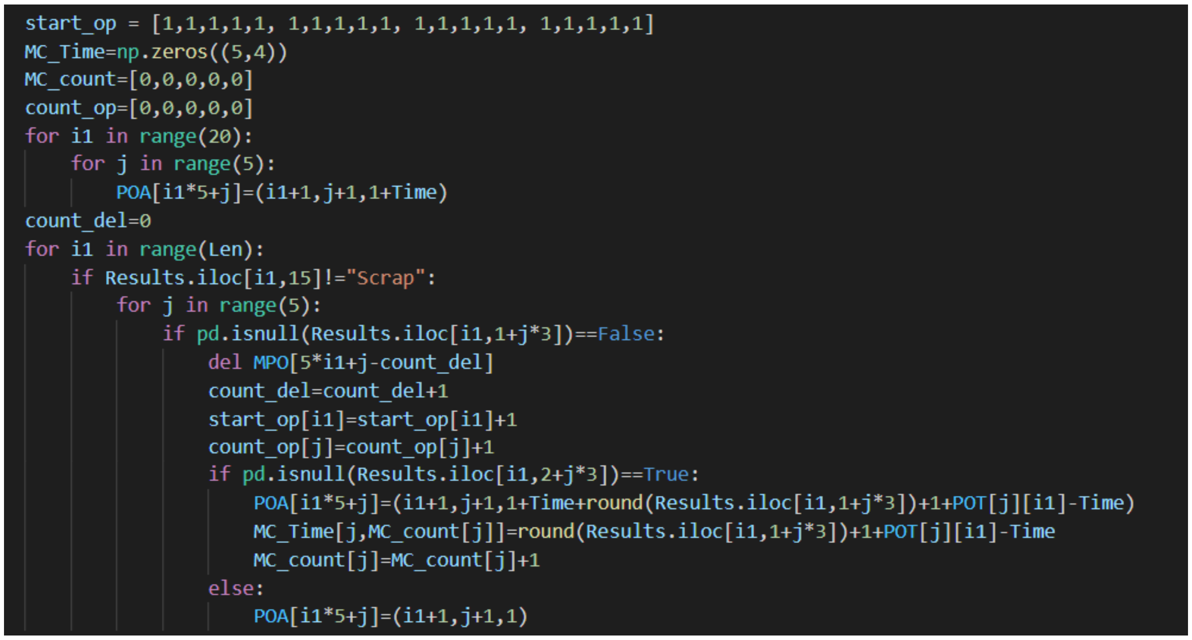
Handle the results data from the previous simulation and get the current production state. Then, recalculate the optimization of process level.



```
#--------------------------------------------------------Part 7
Len=len(Results)
col=Results.shape[1]
i_pro=[0,0,0]
for i1 in range(Len):
    if pd.isnull(Results.iloc[i1,4])==False and pd.isnull(Results.iloc[i1,5])==True:
        if Results.iloc[i1,3]=="Operation 21":
            sr[0]=round(Results.iloc[i1,4]+pt2-Time)
        elif Results.iloc[i1,3]=="Operation 22":
            sr[1]=round(Results.iloc[i1,4]+pt2-Time)
        else:
            sr[2]=round(Results.iloc[i1,4]+pt2-Time)
    if Results.iloc[i1,3]=="Operation 21":
        i_pro[0]=i_pro[0]+1
    elif Results.iloc[i1,3]=="Operation 22":
        i_pro[1]=i_pro[1]+1
    elif Results.iloc[i1,3]=="Operation 23":
        i_pro[2]=i_pro[2]+1
rest=20-i_pro[0]-i_pro[1]-i_pro[2]
```

#Part 8

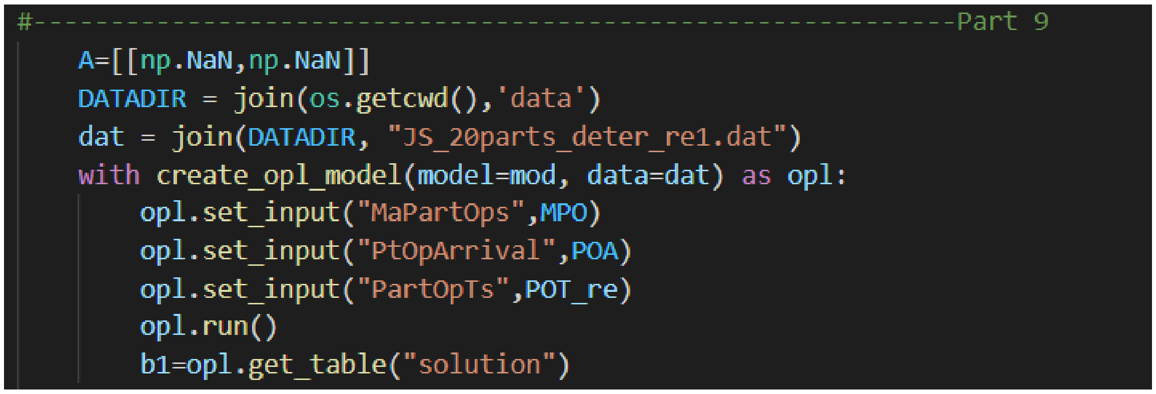
Deal with the data for rescheduling and reschedule the sequence of the parts.



```
start_op = [1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1]
MC_Time=np.zeros((5,4))
MC_count=[0,0,0,0,0]
count_op=[0,0,0,0,0]
for i1 in range(20):
    for j in range(5):
        POA[i1*5+j]=(i1+1,j+1,1+Time)
count_del=0
for i1 in range(Len):
    if Results.iloc[i1,15]!="Scrap":
        for j in range(5):
            if pd.isnull(Results.iloc[i1,1+j*3])==False:
                del MPO[5*i1+j-count_del]
                count_del=count_del+1
                start_op[i1]=start_op[i1]+1
                count_op[j]=count_op[j]+1
                if pd.isnull(Results.iloc[i1,2+j*3])==True:
                    POA[i1*5+j]=(i1+1,j+1,1+Time+round(Results.iloc[i1,1+j*3])+1+POT[j][i1]-Time)
                    MC_Time[j,MC_count[j]]=round(Results.iloc[i1,1+j*3])+1+POT[j][i1]-Time
                    MC_count[j]=MC_count[j]+1
                else:
                    POA[i1*5+j]=(i1+1,j+1,1)
```

#Part 9

Run CPLEX module for rescheduling and deals with the solution. Save the data into Excel file "CPLEX_Solution.xlsx".



```
#-----------------------------------------------------------Part 9
    A=[[np.NaN,np.NaN]]
    DATADIR = join(os.getcwd(),'data')
    dat = join(DATADIR, "JS_20parts_deter_re1.dat")
    with create_opl_model(model=mod, data=dat) as opl:
        opl.set_input("MaPartOps",MPO)
        opl.set_input("PtOpArrival",POA)
        opl.set_input("PartOpTs",POT_re)
        opl.run()
        b1=opl.get_table("solution")
```

Instruction

L150, L186, L395, L405: Set the path of excel file "sequence.xlsx".

L158, L201, L211, L214, L365: Set the path of excel file "sequence1.xlsx".

L166: Set the path of Simul8 model file "Schedule2.s8".

L197, L200: Set the path of excel file "sequence2.xlsx".

L331: Set the path of excel file "data5_re.xlsx".

Set the correct path for each file in the code. Under current path, CESMII folder should be put in D disk. Run the master program. The main program and subprogram can run automatically.

4.4 CPLEX program

Files:

JS_20parts_deter1.dat

JS_20parts_deter_re1.dat

JS_20parts_deter1.mod

Details

"JS_20parts_deter1.dat" is the data file for initial schedule which stores the values of those fixed variables. The variables include:

nbTime: A constant. The total time periods for the whole production process.

nbMachineType: A constant. The number of machine types.

nNbPart: A constant. The number of parts nbOperation: A constant. The number of operations per part

nbPartOpeation: A vector. The number of operations required by each part

PartDue: A vector. The due time of each part.

PartOpTs = A tuple with three columns. They mean part number, operation number and corresponding processing time, respectively.

MaPartOps: A tuple with three columns. They mean machine type number, part number and operation number, respectively.

PtOpArrival: A tuple with three columns. They mean part number, operation number and corresponding earliest arrival time, respectively.

Besides, this data file connects to “data5.xlsx”, in which following variables are stored.

MachineCap: A matrix. Each column means the capacity of each machine and each row means time period.

operation: A vector. The beginning operation of each part.

“JS_20parts_deter_re1.dat” is the data file for reschedule which stores the values of those fixed variables. The variables include:

nbTime: A constant. The total time periods for the whole production process.

nbMachineType: A constant. The number of machine types.

nNbPart: A constant. The number of parts

nbOperation: A constant. The number of operations per part

nbPartOpeation: A vector. The number of operations required by each part

PartDue: A vector. The due time of each part.

PartOpTs = A tuple with three columns. They mean part number, operation number and corresponding processing time, respectively.

Besides, this data file connects to “data5_re.xlsx”, in which following variables are updated via Python mater program.

MachineCap: A matrix. Each column means the capacity of each machine and each row means time period.

operation: A vector. The beginning operation of each part.

“JS_20parts_deter1.mod” is the model file in which the problem formulation and optimization algorithm (Surrogate Lagrangian Relaxation (SLR) algorithm) are coded.