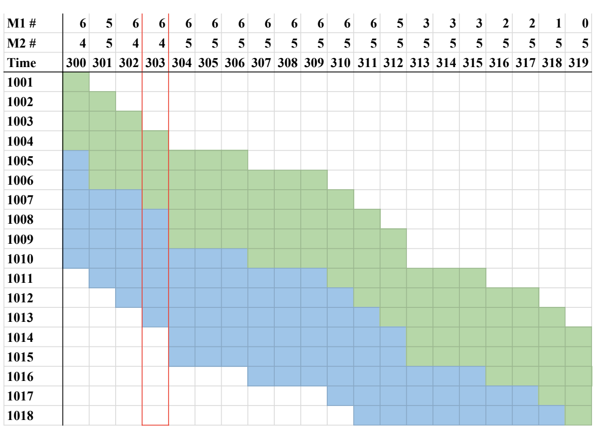
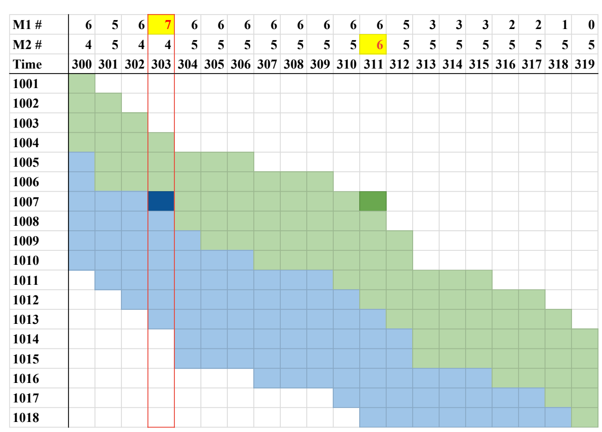
Appendix B. Replanning Procedure

Due to the stochastic nature of processing times and material arrivals, the short-term schedules need to be adjusted periodically to account for orders that were released in previous periods waiting to be started. As discussed in Section 3, the scheduler usually updates the schedules two to three times a week and uses the best-case estimates of the TPDs. For standard product A, the estimates are eight and seven days in the integration and testing stages. Their current replanning scheme employs a heuristic based on machine availability. The main idea is straightforward: whenever there is a suitable available machine and the allowable number of starts per week has not been exceeded, the job with the highest priority will be scheduled to start. Although this might not be the best strategy to follow due to the imbalance of TPDs and the difference in the number of machines assigned to the two processing stages, it is a simple approach that is easily implemented. The current scheme also meets the goal of finishing the most current orders before their due date

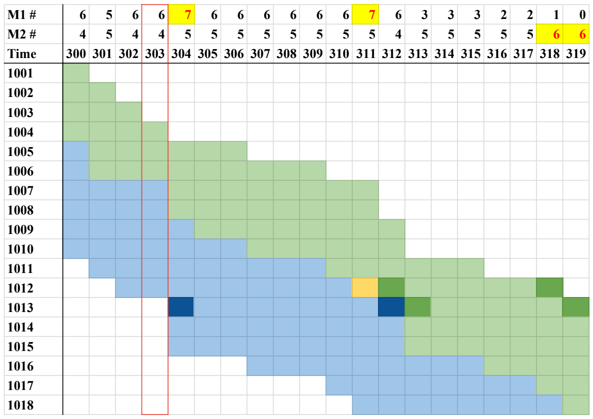
Feedback from progress on the shop floor is considered when replanning. In our simulation model, we continue to use the logic that the company is using in their approach. However, output is checked only once a week and replanning is triggered when any of the essential resources are unavailable. As discussed in Section 4.1, four resources need to be checked before starting a new job. If a job goes into any of the queues in **Error! Reference source not found.** in the main paper, that means current processing is delayed and the corresponding job needs to be rescheduled. Figure B1 depicts an example of the Gantt charts used to record the most current plan. The first and second rows in each figure represent the total number of machines in use in the integration and testing stages. The third row indicates the time period or day in the planning horizon. Job numbers are shown in the leftmost column in order of priority. The green bars (upper portion) represent the processing time in testing stage and the blue bars (lower portion) are the processing time in integration stage. The highlighted cells with darker colors represent the changes made compared with the previous plan. Assuming that 6 machines are available in integration stage and 5 machines are available in testing stage, the red numbers with yellow background in the first and second rows indicate the number of planned jobs that will exceed machine capacity.



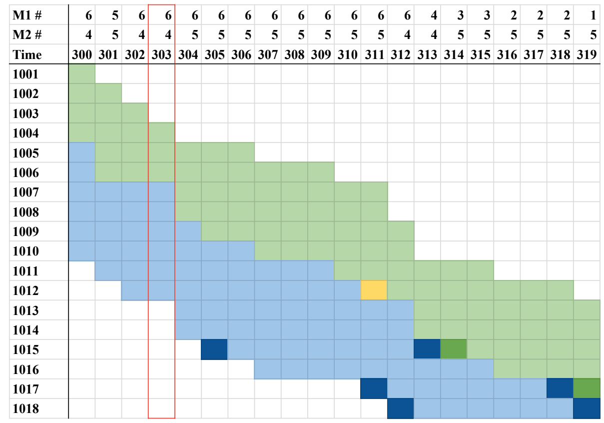
B1a. Original plan



B1b. Trigger event



B1c. First update



B1d. Second update

Figure B1. An example of the current rescheduling logic

Figure B1a depicts the original plan before circumstances trigger a need to replan. We assume that the plan is checked on day 303 (boxed in red). As shown Jobs 1001, 1002, and 1003 have finished, Job 1004 is planned to be finished by the end of the day, and Job 1007 is supposed to complete Integration stage processing at the end of day 302. Job 1013 is a new start planned on day 303. Based on nine days for Integration stage processing and seven days for Testing stage processing, the planned EOP is at the end of day 318. For the given number of machines in each stage, the original plan is feasible since the machine requirements are less than or equal to machine availability.

Figure B1b describes how the delay of integration stage TPDs triggers replanning. In this example, the technicians find that one more day is needed to finish the integration work for Job 1007. This pushes the completion day from 302 to 303 so the planned EOP is shifted one day. Consequently, the schedule has to be replanned because the required number of machines in the first stage is seven while the number available is six. Additionally, due to the shifting of the testing slot of Job 1007, the current schedule is not feasible on day 311 either.

The first update is shown in Figure B1c. Following the heuristic rule discussed above, jobs with the least priority on days where machine capacity is exceeded must be rescheduled to open up slots for those with higher priority. On day 303, Job 1013 has the lowest priority, so it has to be shifted to the next day to make the schedule on 303 feasible. Integration end time, testing start time, and the planned EOP are also shifted accordingly. On day 311, Job 1012 has the least priority for transitioning to testing stage but we cannot change its start day because it already started on day 303. Also, on day 311, since there are already 5 jobs being processed in testing stage, there is no more capacity to start a new job in testing stage. A block, colored in yellow, is the day that Job 1012 stays on the machine in the first stage but is not being processed.

Although the first update removed the infeasibility on day 303 (see Figure B1b), other days became infeasible due to the changes. Highlighted in red number and yellow background in the first and second rows, days 304, 311, 318, and 319 exceed the machine capacity after the first update. Moreover, the starting capacity rule is violated on day 304 since no more than two starts are allowed each day. After additional updates, we arrive at the Gantt chart in Figure B1d which depicts a feasible plan that assures that the higher priority jobs are completed as early as possible.

Finally, note that replanning might also be triggered by a change of the order of job priorities or the insertion of preferred customer orders. Such changes, however, would simply push back release or Planned EOP dates for all jobs already on the schedule and would not affect capacity constraints. Therefore, we have not considered those scenarios when evaluating either short-term or long-term planning strategies.

Appendix C. Example of CT Generation

Figure C1 shows a detailed example of how the CT is generated: The first value is generated as TPDs equals to 12 days, colored in blue, according to the product type distribution. If the backend material is not delayed, the job will start on day 302 and finish on day 313. In the case when back-end material is in shortage, we need to generate parameters *x* which represents the number of days that the job is stopped by the back-end material delay and *y* which represents the number of days after the job starts. Suppose $x = 4$ and $y = 6$ are generated according to the distribution shown in Table 5 in the main paper. The processing is stopped on day 307 of the simulation. Four days later, colored in red, the processing is restarted. The job is finished on day 317 of the simulation.

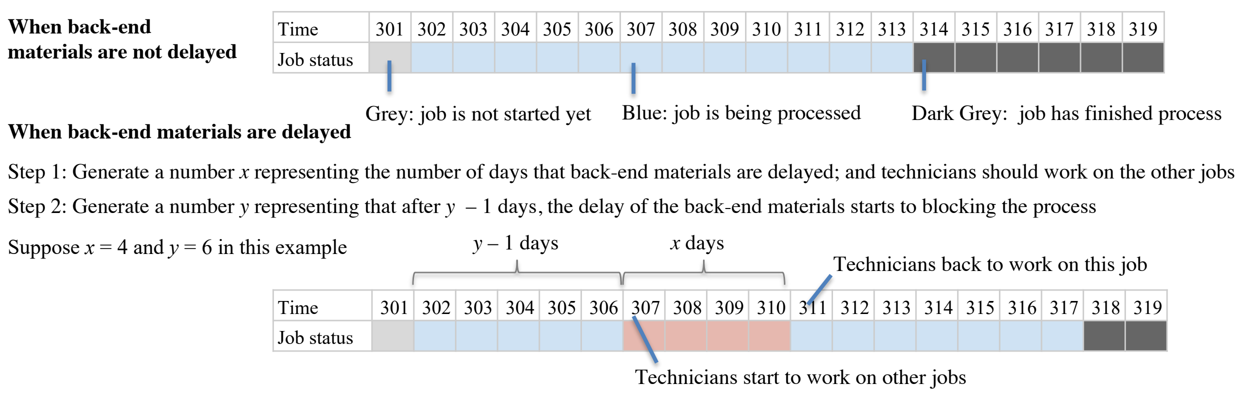


Figure C1. An example of CT generation when back-end material delay

Appendix D. Detailed Simulation Diagram

Figure D1 shows the detailed simulation diagram. There are five main steps in the simulation. The first dotted blue box represents Step 1, start and initialization. The simulation starts by setting the number of repetitions *i* as 0. Before each simulation replication (i.e., rep), *i* will be updated as $i = i + 1$. Similar to rep *i*, simulation day *t* is set as 0 at the beginning of each rep. Then, job and order information are input to the simulation from the MPP file. If a job was under processing at $t = 0$, the exact historical information would be set as job parameters in the system. If a job in the MPP file has not been started yet, the job parameters will be generated at $t = 0$. At the same time, bays' and workers' states are updated. Finally, all the classes, objects, variables, and statistical counters are initialized. For example, parameters such as SLHs, core arrival time, back-end material delay days, and processing times are generated based on the parameters' distribution assumptions. Statistical counters such as the number of throughputs and the number of jobs that fail to finish before the planned time are set as 0. After initializations, *t* will be updated as $t = t + 1$ for the next time period in the simulation.

When it goes to a new week in the second step, i.e., when. t % 6 is 0, the replanning function is applied. After replanning, the new start and corresponding queues statistics should be renewed as empty.

Then, bays' and workers' states are updated. The last task in the second step is to pull jobs planned in the following six days if the core parts for the jobs have arrived. As the resources for the jobs planned at *t* might not be available, we create the flexibilities of the starts by checking jobs six days ahead of the planned time. In addition, `additional' workers as described in Section 4.2 should be set as `idle' since new jobs have higher priority to use the additional workers.

Step 3 checks the integration resource availabilities and updates integration statistics. The step prior to assigning workers to start a new job is to check if back-end materials are delayed and subsequently stop the processing of the jobs. If the back-end materials are delayed, workers processing the delayed jobs should be set as `idle' at the beginning of the day *t* and labor resources should be updated. Then, for each of the jobs pulled from the planning list in Step 1, if any of the four starting conditions we discussed in Figure 2 in the main paper are not satisfied, the job cannot be started and is pushed to the integration queue responding to the first unavailable resource. If all four conditions return `Yes', the new job can be started and bays' and workers' states should be updated from `idle' to `working'. After that, we check if back-end materials for the stopped jobs in the previous period $t - 1$ are available now at *t*. If the condition is `Yes', we check if available workers are in the technician pool and assign them to the stopped jobs. The last time in the integration process to check if there are `idle' state workers is to assign additional workers to the processing jobs to speed up the processing. Note that in the diamond flows, we distinguish the term available and additional workers. When assigning available workers to the job, we assign the standard number of workers in the crew. When it comes to the additional workers, we assign more workers to the processing jobs because there are still workers in the technician pool whose states are `idle'. After knowing the number of workers working on each job in the job processing list, we update the integration process by adding the daily CLHs at t to the total CLHs. If the jobs' total CLHs are greater than or equal to the SLHs, the integration is finished and workers are released at the end of day *t*. Finally, if jobs are finished and testing bays are available, the integration bays are released and jobs are pushed to the testing bays. At the same time, the bays states should be updated.

Step 4 is the testing process. Similar to Step 3, workers' states affected by back-end material delays should be updated before assigning workers to new jobs. Since new jobs are pushed to testing bays in Step 3 already, the only resource that needs to be checked for new jobs is labor. If enough workers are in the `idle' state, they will be assigned new jobs. Moreover, if the stopped jobs that were affected by back-end material delay are able to be processed at *t* and have enough idle workers after new jobs are assigned with workers, the remaining available workers will be assigned to the stopped jobs and workers' states are updated. After that, if more workers are available, they will be assigned to the processing jobs with higher priorities as additional workers to speed up the processing. When jobs' testing is finished, both workers and bays are released.

Step 5 wraps up the simulation information on day *t*. The first task is updating the classes, variables, and statistical counters at the end of the day *t*. This task is important because if the processing is finished in either of the stages at day *t*, the workers should be released at the end of the day *t* and they will be available at the beginning of day $t + 1$. Then, when *t* equals the planned time, the next rep will start and $i = i + 1$. Otherwise, the simulation day moves to the next slot. Finally, when the rep *i* hits the planned rep number, the simulation is stopped for a particular scenario.

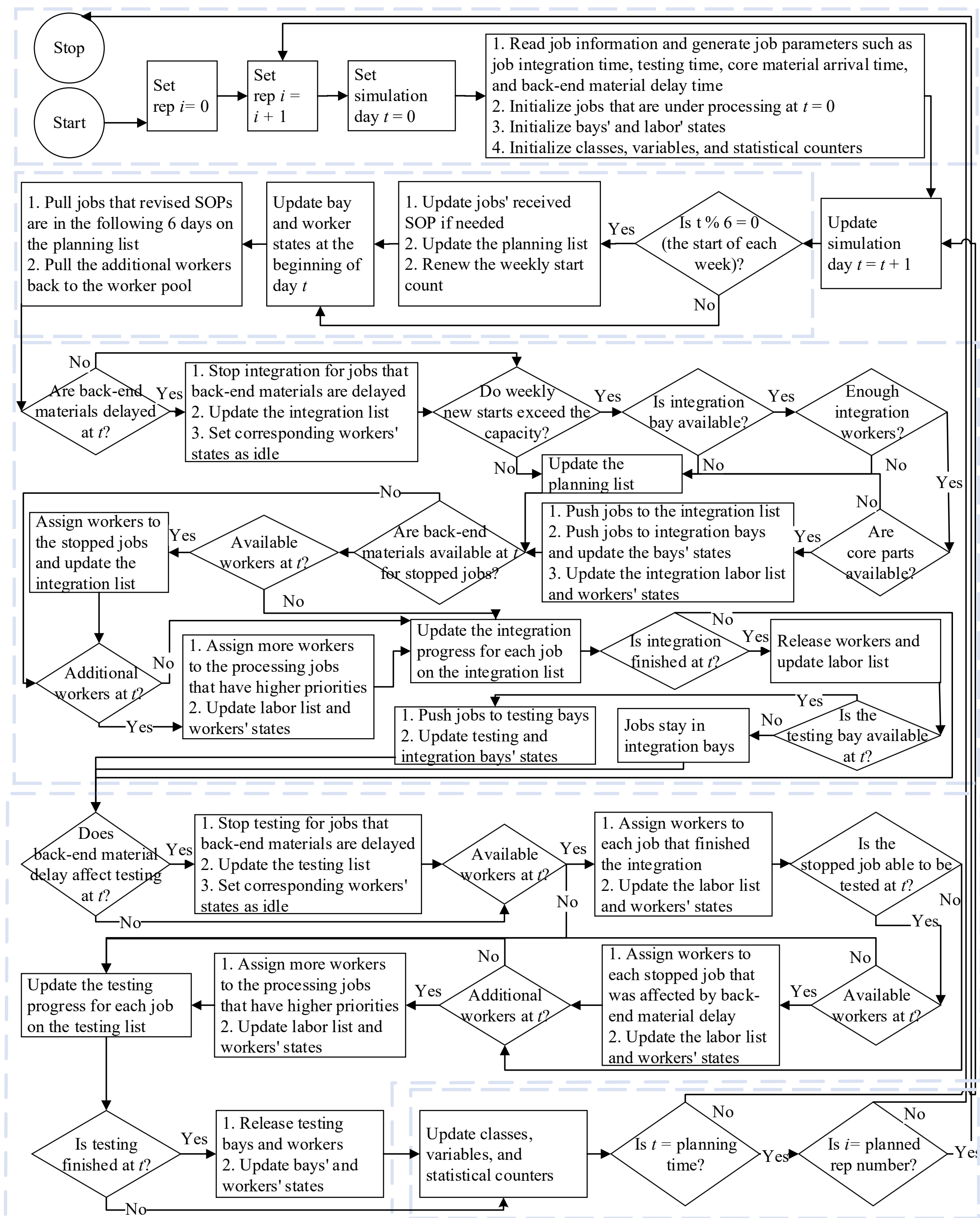


Figure D1. Detailed Simulation Diagram

Appendix E. Detailed OOP-Based Pseudocode

Sets and indices

I	Rep in each set of experiments; $i \in I = \{0, 1, 2, \ldots, 100\}$
T	Simulation Day (in day unit); $t \in T = \{0, 1, 2, \ldots, 1000\}$

Classes

JOB	Product in the processing line; *job* attributes are *Name*, *Number*, *Planned_Start*, *Revised_Start*, *Actual_Start, Planned_EOP*, *Revised_EOP*, *Int_Day*, *Int_Block*, *Int_Delay_Day*, *Int_Delay_Time*, *Int_Delay_On*, *Int_Labor*, *Int_Process*, *Int_On*, *Int_CLH*, *Int_Aactual*, *Test_Day*, *Test_Block*, *Test_Delay_Day*, *Test_Delay_Time*, *Test_Delay_On*, *Test_Labor*, *Test_Process*, *Test_On*, *Test_CLH*, *Test_Aactual*, *Core_Arrival_Time*
IB	Integration Bay; attributes are *Name*, *Jobs*, *Free*, and *Utilization*
TB	Integration Bay; attributes are *Name*, *Jobs*, *Free*, and *Utilization*
IL	Integration Labor; attributes are *Name*, *Jobs*, *Free*, and *Utilization*
IL	Testing Labor; attributes are *Name*, *Jobs*, *Free*, and *Utilization*
QC	Queue where jobs' core parts have not arrived; attributes are *Current* and *Recorded*
QIL	Queue where Standard Product A jobs' core arrived but not enough Integration labor is available; attributes are *Current* and *Recorded*
QIL	Queue where jobs stay on the Testing Bay and wait for Testing Labor; attributes are *Current* and *Recorded*
QI	Queue where jobs' core and labor resources are available, but the integration machine is not; attributes are *Current* and *Recorded*
QS	Queue where jobs cannot be started because the number of new starts hit the capacity; attributes are *Current* and *Recorded*

Variables

PL	Planning List of jobs' release order
SIP	List of Standard Products A in the Integration Process
STP	List of Standard Products A in the Testing Process
SDIP	List of Standard Products A that are delayed in Integration Bay due to back-end material delays
SDTP	List of Standard Products A that are delayed in Testing Bay due to back-end material delays
UIB	List of Integration Bays that are not available for new jobs
AIB	List of Integration Bays that are available for new jobs
UTB	List of Testing Bays that are not available for new jobs
ATB	List of Testing Bays that are available for new jobs
UIL	List of Integration labor who are processing jobs
AIL	List of Integration labor who are not processing jobs
UTL	List of Testing labor who are processing jobs
ATL	List of Testing labor who are not processing jobs
EIL	List of *extra* technicians that are assigned to an Integration job even if that job already has full labor
ETL	List of *extra* technicians that are assigned to a Testing job even if that job already has full labor
MQ	List of jobs that finished the integration process

Statistical Counters

StdS Record the number of new starts of Standard Product A

AIP Record the number of advanced products in processing

CP Number of complete jobs in each category

Four types of Classes were mostly used in the development of the OOP simulation model. The first type of Class has a list of job Objects that are generated at the beginning of the simulation (Day 0). Attributes of the jobs listed in the above notation of the *JOB* Class. The *Int_Day* represents the essential days required when technicians are processing the integration tasks for a job. *Int_Day* is TPDs we introduced before and is different from *Int_Avtual*. *Int_Avtual* is the total number of days that a job stays on the machine, including when jobs are blocked in the integration bay when no testing is available (*Int_Block*) and when no technicians are processing the job because the back-end material delays (*Int_Delay_Day*). Therefore, *Int_Actual* = *Int_Day* + *Block_Day* + *Int_Delay_Day*. We also introduced the *Int_Delay_Time* to describe the start day that back-end materials are delayed, i.e. the day that job's Integration is stopped. More assumptions and distributions of *Int_Delay_Day* and *Int_Delay_Time* are explained in Section 6. *Int_Delay_On* is a Counter that records the total number of days a job has been delayed. Once *Int_Delay_On* is the same as *Int_Delay_Day* at Day t, the materials arrive and labor will be pulled back to work on the job. *Int_Process* is similar to *Int_Day* but in the hour's unit. *Int_Process* is the Total Expected Required Labor Hour (Total ERLH) generated from the job category prediction. As described in Assumption 5, we have considered the LIF and Total ERLH = SLH ÷ LIF. *Int_On* counts the total CLHs in Stage 1 accumulated on a certain job. Once *Int_On* ≥ *Int_Process*, the integration is fished and labor is released to open to other jobs. *Int_Labor* is a list of labor names that record the technicians working on the job each day. Once the integration starts, *Int_Labor* starts to record until the integration is finished. If no technicians are working on the job due to material delays, the *Int_Labor* will record a 0 on that day. *Int_CLH* is the contribution of labor hours of a given job on a given day in Stage 1. As we recorded the number of technicians working on each job each day, *Int_CLH* will be updated based on the number of workers and uses the information in Table 2.

Testing dates are similar to Integration dates. The only difference is that the *Test_Block* happens before technicians start working on the Testing tasks while *Int_Block* happens after Integration starts. As we have more bay capacity than labor capacity, the job might be pushed into the Testing Bay even if not enough labor is available. In this case, the Testing process is blocked due to the labor shortage. For jobs that are not Standard Product A, the labor resource is not considered; and the attributes related to labor are initialized as *None*. Details on the processing time generation under different scenarios are discussed in Section 6.

The second type of Class has a list of Bay Objects. As there are two stages in the studied problem, two Classes are created to represent different processing machines. The most important attribute of the bay is *Free*, a list of dates that record the release time of the bays. The *Free* list is always nonempty as the first element is the date that the bay is introduced to the system (i.e., the first date that the bay is released). For most bays, the first element is 1, representing the first day of the simulation because they are introduced to the system at the beginning of Day 1. Some bays are introduced to the system later. The first elements for them in the list are subsequently not 1. For example, we know that one Testing Bay is introduced to the system at the beginning of 2022, Day 354 of the simulation, the *Free* list of the new bay would be [354] in the time before Day 354. More dates will be appended to the list if the bay processes the other jobs and the release time is estimated. The *Name* is a string that distinguishes different bays. *Jobs* attribute is a list of job numbers that records the jobs that have been processed on the bay. Finally, *Utilization* is a local variable in the Bay Class that records the utilization of bays over time. The third type of Class, related to labor, is very similar to the second type as they are both resources.

The fourth type of Class is a Queue of jobs waiting for specific resources. Note that whenever jobs are pushed into the queue, the queue will be sorted by the jobs Planned EOP. The first element in the queue has the highest priority to pull the resources when available. Two attributes are in the Queue Class. *Current* attribute records a list of jobs over time. *Recorded* attribute is a list of numbers that represent the Queue length at the end of each Day. The length of *Recorded* list should be the same as the simulation planning time as new elements are added to the list very day.

Variables are lists of Objects from Classes. Statistical Counters are lists of numbers. Both Variables and Statistical Counters are a function of (i, t). For example, in each simulation experiment, *SIP*(i, t) represents a list of Standard Product A jobs under the Integration Process at Day t in Rep i. They should be subsequently updated at each Day in each Rep. *SIP*(i, t) and *STP*(i, t) distinguish the Standard Product A jobs from the other jobs. All elements in the two lists should be Standard Product A and those jobs are processed in the Bay at Day t in Rep i. *SDIP*(i, t) records jobs staying on the Integration Bay and waiting for back-end materials to arrive to continue the Integration Process by technicians. The sum number of the jobs in *SIP*(i, t) and *SDIP*(i, t) represents the total number of Integration Bays occupied by Standard Product A at Day t in Rep i. *SDTP*(i, t) contains Testing jobs staying on the Testing Bay and waiting to be processed by Testing Labor at Day t in Rep i. Jobs in both *SDIP* and *SDTP* stay in the Bay but are not processed by technicians. For all the resources, there is one list recording the available resource at Day t in Rep i and another list recording the used resource. For example, elements in *AIB* are the Integration Bays that are available for new jobs. Elements in *UIB* are the Integration Bays that are processing jobs or not processing but holding jobs because no Testing Bay available. *EIL* and *ETL* are the lists that record *extra* technicians in any given Day. Since the priority of labor is to start more new jobs, the extra technicians are required to leave their current jobs when new jobs are available to begin.

Finally, we introduce three Statistical Counters, *StdS*(i, t), *AIP*(i, t), and *CP*(i, t), to record the number of Standard Product A new starts, the number of advanced products in process, and the total number of completed jobs at Day t in Rep i. *StdS* will be set as 0 every six days to count the new starts every week. One will be added to *AIP* if an advanced product starts Integration. Vice versa, one will be deducted from *AIP* if the product finishes Integration.

Function 1. Initialization

Input: Classes, Variables, Statistical Counters at Day 0 in Rep i; MPP file

***Step* 1**: Job order initialization

For *JOB* in MPP file {

If *JOB.Planned_Start* < 1{

The job's features are the same as true historical data because they started before the simulation starts

} ***Else*** {

Generate job features:

If *JOB.Name* is not Standard Product A {

Generate *JOB.Int_Day*, *JOB.Test_Day*, and *JOB.Core_Arrival_Time*;

Let *JOB.Planned_SOP*, *JOB.Revised_SOP* = Planned SOP from the MPP file

Let *JOB.Planned_EOP*, *JOB.Revised_EOP* = Planned SOP from the MPP file

Let *JOB.Actual_SOP*, *JOB.Actual_EOP = None*

Let *JOB.Int_Delay_Day*, *JOB.Int_Delay_Time*, *JOB.Int_Delay_Time*, *JOB.Int_Delay_On*, *JOB.Int_Labor*, *JOB.Int_Process*, *JOB.Int_On*, *JOB.Int_CLH*, *JOB.Int_Delay_Day*, *JOB.Test_Delay_Day*, *JOB.Test_Delay_Time*, *JOB.Test_Delay_Time*, *JOB.Test_Delay_On*, *JOB.Test_Labor*, *JOB.Test_Process*, *JOB.Test_On*, *JOB.Test_CLH*, *JOB.Test_Delay_Day, JOB.Test_Block = None*

JOB.Int_Block = 0

} ***Else*** {

Generate job type and let *JOB.Int_Process* and *JOB.Test_Process* be the Total ERLH for that type

Generate *JOB.Int_Delay_Day*, *JOB.Int_Delay_Time*, *JOB.Test_Delay_Time*, *JOB.Test_Delay_Day*, and *JOB.Core_Arrival_Time*

Let *JOB.Int_Day*, *JOB.Int_Day_On*, *JOB.Int_On*, *JOB.Int_CLH*, *JOB.Int_Actual*, *JOB.Int_Block*, *JOB.Test_Day*, *JOB.Test_Day_On*, *JOB.Test_On*, *JOB.Test_CLH*, *JOB.Test_Actual*, *JOB.Test_Block* = 0

}

}

}

Step **2**: Machine and Labor initialization

For *Resource* in [*IB*, TB, *IL*, *TL*] {

Resource.Utilization = [ ]

For *Object* in *Resource* {

Object.Free = [*object_introduce_time*]

Object.Jobs = [ ]

Object.Name = *Resource_Name*

Append *Object.Name* into the Used_Resource list

}

}

Step **3**: Initialize Variables and Statistical Counters

For *variable* in *Variables* {

variable = [ ]

}

For *c* in *Statistical_Counters*{

c = 0

}

***Output*:** Classes, Variables, Statistical Counters at Day 0 in Rep *i*

Function 2. Integration

```
Input: Classes, Variables, Statistical Counters, Day t, and Rep i
Step 1: Replanning
If the start of each week (t % 6 is 0) {
    Let StdS = 0
    If there are any jobs in the QI, QS, and QIL {
        Replan and update PL, QI, QIL, and QS
        Update jobs Revised_SOP and Revised_EOP
    }
}
Step 2: Update Machine state and job completion data
For IB in UIB {
    If the value of the last element in the IB.Free ≤ t{
        Pull the IB from UIB and push it to AIB
        If JOB in the bay just finished the Integration process {
            Let JOB.Int_Actual = JOB.Int_Day + JOB.Int_Delay_On + JOB.Int_Block
            If JOB.Name is Standard Product A {
                Remove JOB from SIP
            }
            If JOB.Name is Advanced Product A or Advanced Product B {
                Let AIP = AIP – 1
            }
            Push JOB into MQ
        }
    }
}
For TB in UTB {
    If the value of the last element in the TB.Free ≤ t{
        Pull the TB from UTB and push it to ATB
        Update job just finished Testing process as JOB.Test_Actual = JOB.Test_Day +
        JOB.Test_Delay_On + JOB.Test_Block
        Let JOB.Actual_EOP = t
        Add one completion to CP in the corresponding job type at t
        If JOB.Name is Standard Product A {
            Remove JOB from STP
        }
    }
}
Step 3: Update Labor state
For IL in UIL {
    If the value of the last element in the IL.Free ≤ t{
        Pull the IL from UIL and push it to AIL
```

```
        }
}
For TL in UTL {
        If the value of the last element in the TL.Free ≤ t{
                Pull the TL from UTL and push it to ATL
        }
}
```

***Step* 4**: Check if Back-end materials affect the labor assignment

Check Integration Labor:

```
For JOB in SIP {
        If JOB.Int_Delay_Time  = JOB.Int_On + 1{
                Pull JOB from SIP to SDIP
        }
}
For JOB in SDIP {
        Let JOB.Int_Delay_On = JOB.Int_Delay_On + 1
        If JOB.Int_Delay_On ≥ JOB.Int_Delay_Day {
                If there is enough labor in the combination list of AIL and EIL {
                        Assign four labor to JOB
                        Update JOB.Int_Labor and JOB.Int_CLH
                        For Labor assigned to JOB {
                                Update Labor.Free and Labor.Jobs
                                If Labor in EIL {
                                        Remove Labor from EIL
                                        Let J represent the last element in Labor.Jobs
                                        Update J.Int_Labor and J.Int_CLH
                                        Let IB be the Integration Bay the Labor was working on
                                        Update IB.Free
                                } Else {
                                        Pull Labor from AIL to UIL
                                }
                        }
                }
        }
}
```

Check Testing Labor:

```
For JOB in TP {
        If JOB.Test_Delay_Time = JOB. Test _On + 1{
                Pull JOB from TP to SDTP
        }
}
For JOB in SDTP {
```

Let *JOB. Test_Delay_On* = *JOB.Test_Delay_On* + 1
If *JOB. Test_Delay_On* ≥ *JOB. Test_Delay_Days* {
If there is enough labor in the combination list of *ATL* and *ETL* {
Assign two labor to *JOB*
Update *JOB. Test_Labor* and *JOB. Test_CLH*
For *Labor* assigned to *JOB* {
Update *Labor.Free* and *Labor.Jobs*
If *Labor* in *ETL* {
Remove *Labor* from *ETL*
Let *J* represent the last element in *Labor.Jobs*
Update *J.Test_Labor* and *J.Test_CLH*
Let *TB* be the Testing Bay the Labor was working on
Update *TB.Free*
} ***Else*** {
Pull *Labo*r from *ATL* to *UTL*
}
}
}
}
}
***Step* 5**: Update start order and check Integration resources
Bench Build Resource (new start capacity):
For each *JOB* from the *PL* from today to the following 6 days {
If *StdS* ≥ *Start_Capacity* {
Pull the remaining standard jobs from *PL* to the *QS*
}
If *AIP* ≥ *Planned_Advanced_Product_Capcity* {
Pull the remaining advanced jobs from *PL* to the *QS*
}
Pull the *JOB* from *PL* and push it to *QC*
}
}
Core Part Resource:
For *JOB* in *QC* {
If *JOB.Core_Arrival_Time* ≤ *t* {
If *JOB.Name* is Standard Product A {
Pull *JOB* from *QC* to *QIL*
}
Pull *JOB* from *QC* to *QI*
}
}
Labor Resource:

```
For JOB in QIL {
    If enough Integration Labor {
        StdS = StdS + 1
        Assign four Integration Labor to the JOB
        Pull those Labor from AIL to UIL
        Update JOB.Int_Day, JOB.Int_Labor, JOB.LCH,
        Append JOB.Number to and each Labor.Jobs list
        Pull JOB from QIL to QI
        For Lab in JOB.Int_Labor {
            Append (t + JOB.Int_Process ÷ JOB.LCH) to Lab.Free
            Append (JOB.Number) to Lab.Jobs
        }
    }
}
Bay Resource:
For JOB in QI {
    If the corresponding Integration Bay is available {
        Pull IB from AIB to UIB
        Let JOB.Actual_SOP = t
        Append (t + JOB.Int_Day) to IB.Free
        Append JOB.Number to IB.Jobs
        If JOB.Name is Standard Product A {
            Pull JOB from QI to SIP
        }
        If JOB is an advanced product {
            Let AIP = AIP + 1
        }
    }
}
Step 6: Check if jobs are blocked in the Integration Bay
For JOB in MQ {
    Denote the Bay that JOB stays on at the beginning of Day t as IB
    If any suitable TB is available {
        If JOB.Name is Standard Product A {
            Pull JOB from MQ to QTL
            Append JOB to TB.Jobs
        } Else {
            JOB stays on the IB
            Remove JOB from MQ
            Update IB.Jobs and IB.Free
        }
    } Else {
```

```
            Pull JOB from MQ
            Push JOB back to its original Integration Bay
            Update IB.Free
            Update AIB and UIB
            Update JOB.Int_Block and JOB.Int_Actual
        }
}
```

Step 7: Assign Extra Labor to jobs that are under Integration Process

```
For JOB in SIP {
    If AIL is not empty {
        If can assign more Labor to JOB {
            Append JOB.Number to Labor.Jobs
            Update JOB.Int_Labor, JOB.Int_CLH, JOB.Int_Day, and JOB.Int_On
            Denote the Bay that JOB stays on at the beginning of Day t as IB
            Update IB.Free
            For Lab in JOB.Int_Labor {
                Update Lab.Free
            }
        }
    } Else {
        Break for loop
    }
}
```

Output: Classes, Variables, and Statistical Counters

Function 3. Testing

***Input*:** Classes, Variables, Statistical Counters, Day *t*, and Rep *i*

***Step* 1**: Check if Testing Labor is available

```
For JOB in QTL {
    If there is enough available testing labor in either ATL, ETL, or the combination of both list {
        Assign two Labor to JOB.Test_Labor
        Update JOB.Test_Labor, JOB.Test_CLH, and JOB.Test_Day
        For TL in JOB.Test_Labor {
            Update TL.Jobs and TL.Free
            If TL in ETL {
                Let J be the original job that TL is assigned to
                Update J.Test_Labor, J.Test_CLH, and J.Test_Day
                Let TB denote the Testing Bay J stays on
                Update TB.Free
                For Lab in J.Test_Labor {
                    Update Lab.Free
                }
```

} ***Else*** {
Pull *TL* from *ATL* and push it to *UTL*
}
}
} ***Else*** {
Update *JOB.Test_Block*
}
Let *TB* denote the Testing Bay the job stays in
Update *TB.Free*
}
***Step* 2**: Assign Extra Labor to jobs
For *JOB* in *TP* {
If *ATL* is not empty {
If can assign more *TL* to *JOB* {
Append *JOB.Number* to *TL.Jobs*
Update *JOB.Test_Labor*, *JOB Test_CLH*, *JOB.Test_Day*, and *JOB.Test_On*
Let *TB* denote the Testing Bay that is processing *JOB*
Update *TB.Free*
For *Lab* in *JOB. Test_Labor* {
Update *Lab.Free*
}
}
} ***Else*** {
Break for loop
}
}
***Step* 3**: Update job variables at the end of Day t
For *JOB* in *SIP* {
$JOB.Int\_On = JOB.Int\_On + JOB.Int\_CLH$
If $JOB.Int\_On \geq JOB.Int\_Process$ {
Update *JOB.Int_Actual*
Remove *JOB* from *SIP*
}
}
For *JOB* in *STP* {
$JOB.Test\_On = JOB.Test\_On + JOB.Test\_CLH$
If $JOB.\ Test\_On \geq JOB.\ Test\_Process$ {
Update *JOB. Test_Actual*
Remove *JOB* from *STP*
}
}

***Step* 4**: Update Resource Utilization

$IB.Utilization(t)$ = (length of $UIB(t)$) ÷ (length of $UIB(t)$ + length of $AIB(t)$) × 100%

$TB.Utilization(t)$ = (length of $UTB(t)$) ÷ (length of $UIB(t)$ + length of $ATB(t)$) × 100%

$IL.Utilization(t)$ = (length of $UIL(t)$) ÷ (length of $UIL(t)$ + length of $AIL(t)$) × 100%

$TL.Utilization(t)$ = (length of $UTL(t)$) ÷ (length of $UIL(t)$ + length of $ATL(t)$) × 100%

***Step* 5**: Update Queues

$QC.Recorded(t)$ = length of QC

$QI.Recorded(t)$ = length of QI

$QIL.Recorded(t)$ = length of QIL

$QS.Recorded(t)$ = length of QS

Output: Classes, Variables, and Statistical Counters