TPK4186

Advanced Tools for Performance Engineering

Assignment 3

Herman Zahl & Hans Erik Heum

NTNU

Kunnskap for en bedre verden

Table of Contents

Structure of files **3**

Introduction **3**

Task 1 **3**
- class Batch 3
- class Buffer 4
- class Task 4
- class Machine 4
- class Plant 5
- Printer 6
- Implementation of the plant 6

Task 2 **7**
- class Event 7
- class Schedule 8
- class Simulator 9
- Printer 10

Task 3 **12**
- Class Optimizer 12
- Printer 12

Further explanation and discussion 13

Structure of files

All tasks located in the same python file, WaferProductionTest.py
Another HTML printer is added in additionalHTMLprinter.py for fun. This is commented on later

Introduction

This assignment is structured into tasks, classes and functions. We will begin by explaining the functions we have made for each task, excluding the obvious ones like getCode(), removeBatch(), isEmpty() etc, which, in our opinion, are strongly self-explanatory at this point in the course. In addition, they are of limited interest to the client.

Furthermore, we will try to explain the walkthrough of what actually happens when we create objects, or start simulations. And most importantly, some reasoning on different choices made along and the tests performed.

Task 1

In this task we are asked to create classes and functions to be able to simulate the fabrication process. This is done by making several data structures, making it possible to describe each category of objects involved in the production process.

class Batch

The batch object has constants that define the state of a batch. The batch can not have been loaded into the simulation yet, can be in a buffer, can be in a machine or in the finished inventory. This is described through the variables:

```
    BEFORE_JOINING_BUFFER = 0
    IN_BUFFER = 1
    IN_MACHINE = 2
    FINISHED_IN_INVENTORY = 3
```

__init__(numberOfWafers, Code) - The Batch object must be initialised with the number of batches it contains as well as a unique code to identify the batch. When a batch object is initialized its state is also chosen to be BEFORE_JOINING_BUFFER.

class Buffer

__init__(name, capacity, machine) - The buffer object must be initialized with its own name, the capacity of wafers it can hold and which machine the buffer belongs to. When the Buffer object is made, an empty list called ‘batchesInBuffer’ is also made, which will later serve to store batches that are in this buffer.

getNextBatchInQueue() - The list of batches in this buffer should be handled as a First-In-First-Out (FIFO) queue. Therefore, this function returns the batch that was first added in the buffers queue of batches. FIFO principle is also supported by **AddBatch()** which simply appends the batch to the queue.

hasSpaceForAnotherBatch(newBatch) - This function has a potential new batch as its input. This function iterates through all the batches that are already stored in this buffer, and checks whether the buffer has enough capacity to store the new batch. In that case it returns True, otherwise False.

class Task

__init__(name, machine, inputBuffer, outputBuffer, loadingTime, processingTime) - The task object must be initialized with the name of the task, which machine that does the task, what the inputBuffer is, which buffer the batch is sent to after the task is done, the time it takes to load and unload batches and the time it takes to process on each wafer.

Notice that these two have been combined into one variable ‘loadingTime’ since they are always the same in the simulation.

class Machine

The machine object has constants that describe the logic the machine uses to select which task to operate first.

```
CHRONOLOGICAL_TASK_SELECTION = 1

REVERSE_CHRONOLOGICAL_TASK_SELECTION = 2

CHOOSE_BUFFER_WITH_MOST_BATCHES = 3
```

__init__(name) - The machine object must be initialized with its name. When a machine is created, a list of tasks, buffers and a state-variable that describe what batch the machine is working on is created. The latter is initialized with None, since the machine is not working on any batches when it is made. The list of buffers and batches will later on be extended when buffer and batch objects are made.

selectNextBatch(logic) - This function has the logic which the machine should use as input. It then starts a new function, which finds the next batch to work on, depending on the logic (Chronological selection etc.)

chronologicalTaskSelection() - Since the list of tasks are already chronologically added to the Plant, this function iterates through the list of tasks and checks for each task if it is possible to do the task. That means, for the first machine, we check if task 1 can be done. If not, task 3 is checked, then 6, lastly 9.

reverseChronologicalTaskSelection() - As the name states, this does the same as the function above, just in the opposite order. For the first machine, start with task 9… up to task 1.

chooseBufferWithMostBatches() - This function makes a list of tasks, sorted by the task that has the most batches in its buffer first. Then it iterates through each task and checks it is possible to do the task.

makePriorityListOfBuffers() - This function creates a priority queue where tasks are sorted by the number of batches they have in their buffer. It iterates through each task and inserts the task in a list of tasks, depending on how many buffers the task has in its queue.

We find it reasonable to test this operational policy since it seems natural to start working on potential bottle-necks as soon as possible. It is also important to test different operational policies when our simulator introduces new batches as soon as possible in the first task buffer. A policy of Chronological-selection may then hamper progress throughout the plant. From a practical point of view, this increases the possibility of high levels of WIP (work-in-progress) goods which we often want to minimize (at least keep low).

checkIfTaskCanBeDoneAndReturnBatch(task) - This is the function that all the other logic functions use to check if the task is implementable. For each task it checks if there are any batches in the inputBuffer. If there are none, the operation can not be made and it returns None. It then checks if the possible batch is small enough to fit in the tasks output buffer. If the output buffer does not have enough space, the task can not be done - and None is returned.

class Plant

__init__(name) - When the plant is initialized it must have a name. In addition, dictionaries to contain buffers, tasks and machines are made (with their names as keys). A list to store batches is also made.

newBatch(numberOfWafers, code) - This function makes a new batch object and appends it to the plants list of batches.

newBuffer(nameOfBuffersTask, capacity, machine) - This function makes a new buffer object and stores it in the plant’s dictionary of buffers. This function also adds the buffer to the corresponding machine. The nameOfBuffersTask variable is the task that has this buffer as its input buffer.

newTask(name, machine, outputBuffer, loadingTime, processingTime) - This function creates a new task, adds the task to the corresponding machine and adds the task to the plant's dictionary of tasks. As stated before, loadingTime into and out of machines are the same.

newMachine(name) - This function makes a new machine object and stores it in the plants dictionary of machines.

findFirstTask() - Later on, when the batches enter the first buffer, we use this function to find out which task the batch needs to do first.

The following functions are only used in task 2, but described here regardless.

batchEntersBuffer(batch, buffer) - This function is used when a batch enters a buffer. It sets the state of the batch to IN_BUFFER, so we know it is in a buffer. If the name of the buffer we want to add the batch is the last one (finished inventory) we set the state of the batch to FINISHED_IN_INVENTORY - to show it is finished. The batch is also added to the buffers queue of batches.

loadMachineFromBuffer(batch, buffer, machine) - This function changes the machine WorkingOnBatch-variable to the corresponding batch - to show that it is working on a batch. It also changes the state of the batch, to show that it is now in a machine. The batch is also removed from the corresponding buffer.

batchFinishedOperationOnMachine(machine) - This function changes the batch the machine is working on to None, since it is finished working on a batch

resetPlant() - When we later on are doing several simulations with different logics to optimize the wafer production, this function is used to reset the plant. This is done by resetting each machine and buffer. By resetting, we mean that no machines are working on any batch and all buffers are empty! The resetting is done by iterating through all machines and all buffers and using **resetMachine()** and **resetBuffer().**

Printer

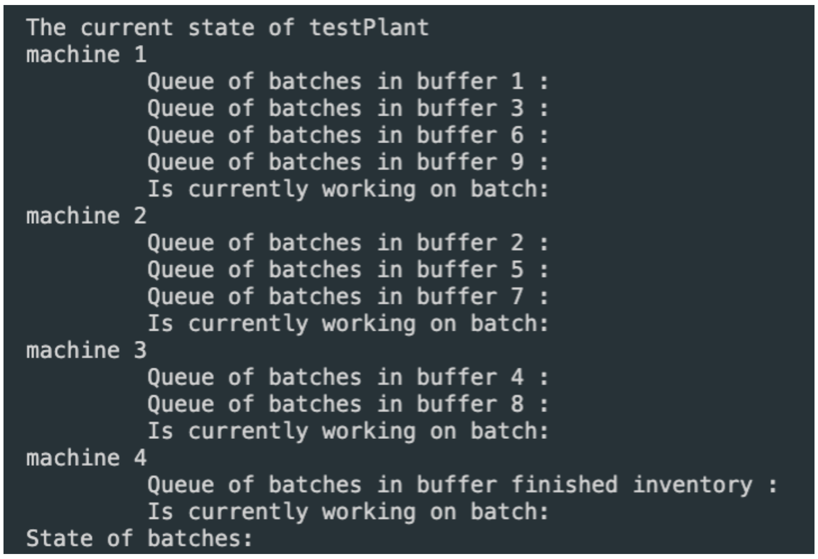
printPlantState(outputFile) - This function serves to show the state the plant is currently in. It shows all the machines in the plant, the machine's buffers and the batches that are currently in the buffers queue. It also shows the batches that each machine is currently working on, and the state of all the batches. The function iterates through all the plants machines, and then iterates through all the machines buffers. This function can be used during a simulation to show the exact state of the plant at this time, or it can be used before any simulations has started to show how the plant looks (picture below)

Implementation of the plant

First, the plant object is initialized. Furthermore the machines are made, with the plants newMachine function. To be able to connect the last "finished inventory" buffer, a last machine is made - which can be interpreted as a machine of finished goods.
Additionally, all the buffers are made. They have a connection to a machine. Here we also make a "final buffer" with infinite capacity, to store the batches that are finished operating. Moreover tasks are mode, which all have a connection to the machine that does the task and its input and output buffer. The tasks are also made with loading time and processing time, retrieved from the assignment.

Task	1	2	3	4	5	6	7	8	9
Time	0.5	3.5	1.2	3	0.8	0.5	1	1.9	0.3

After we have created the Plant, we can use the Printers printPlantState function. Here is how the plant looks like, without creating any batches.



```
The current state of testPlant
machine 1
        Queue of batches in buffer 1 :
        Queue of batches in buffer 3 :
        Queue of batches in buffer 6 :
        Queue of batches in buffer 9 :
        Is currently working on batch:
machine 2
        Queue of batches in buffer 2 :
        Queue of batches in buffer 5 :
        Queue of batches in buffer 7 :
        Is currently working on batch:
machine 3
        Queue of batches in buffer 4 :
        Queue of batches in buffer 8 :
        Is currently working on batch:
machine 4
        Queue of batches in buffer finished inventory :
        Is currently working on batch:
State of batches:
```

Task 2

In this task we implement a discrete event simulator of the fabrication process.

class Event

The Event object represents incidents of importance, that we need to keep track of in the simulation. The Event objects are created and used by both the Schedule object and the Simulator object.

The class object consists of the following constants, that describe the event. Notice how we have chosen to add the additional states 'Machine operates on bach' and 'finished'. This is because we find this more logical, to have a time schedule for when a machine loads a batch, operates on a batch and unloads a batch - all separate events. The simulation will still operate as we just had the events 'load buffer' and 'load machine'. In other words, we differ somewhat from the recommendations in the assignment (only having two states), but after consultancy in exercise lectures and internal discussion, we found this to be a more clear and easier description of the system, that gave us a better grasp of the simulation.
Below is the Events we ended up with:

```
    BATCH_ENTERS_FIRST_BUFFER = 0

    BATCH_ENTERS_BUFFER = 1
    MACHINE_FINISHED_OPERATING_ON_BATCH = 2
    LOAD_MACHINE_FROM_BUFFER = 3
    FINISHED = 4
```

__init__(type, number, date) - the Event object must be initialized with a type, which is one of the constants mentioned above. It must also have a unique number to represent this event. When several events are created this number increases each time. The date when the event is happening must also be included. In addition, variables to store the events batch, machine and task are created.

class Schedule

The schedule object schedules the events that are going to happen. The most important thing in the schedule is the time dimension: events are ordered by what time they are supposed to happen. It calculates the time it takes to process and load batches, and makes new events that appends to the list of things that are going to happen.

__init__(type, plant) - The schedule object must be connected to a plant. A list that is going to hold the events in the schedule is also made. The current date is set to 0 in the beginning. An empty list of batches is also created. This is later going to store all the batches that have been created, but not yet pushed into the system. The variable EventNumber is to keep track of how many events we have created when we make new events. We have also made an additional 'testHandCodedEventNumber' that is used when we start the simulation by creating events coded by hand.

insertEvent(event) - This function has an event as input, and finds out where the event should be placed in the schedule, which is always sorted in ascending order based on the events dates.

scheduleEvent(type, date, batch, task) - This function makes new events, sets the events batch and task to the input values, and inserts the event in the schedule, with insertEvent().

scheduleEventBatchEntersFirstBuffer() - As the name says, this function makes new schedules when a batch is pushed into the simulation. How and when this function is called is crucial to the overall system development because it is the basis for the batch introduction-logic. If there are any batches available that have not yet been pushed into the system, this function pops the first batch in the queue, connects it to the first task and makes a new event that is going to push the batch into the first buffer. The date when the batch is pushed into the new buffer depends on the tasks loading time.

scheduleEventBatchEntersBuffer(batch, task) - This function does the same as one above, just without popping a batch, since it has a batch as input. It finds the loading time based on the task, and schedules an event that the batch should enter this buffer.

scheduleEventLoadMachineFromBuffer(batch, task) - At first, this function changes the state of the machine that is going to load the batch, to show that the machine is operating on a buffer. This also makes the machine 'unavailable', when we later check if any machines can work on something. The date of this event depends on the current date, and the time it takes to load the batch into the machine.

scheduleEventMachineFinishedOperatingOnBatch(batch, task) - When we schedule this event, the date depends on the time it takes to process all the batches. The new event is of type 'MACHINE_FINISHED_OPERATING_ON_BATCH', to show that the process is finished. As stated earlier, this extra event could be excluded, but we thought it looks better to actually separate all the events.

class Simulator

This class represents the simulation of batches going through the system. The start of the simulations and the executions of all the events are handled in this class.

The class has constants, which represents the logic we have chosen to introduce new batches into the system. The only batch introduction-logic is to load new batches when there is capacity at the first buffer.

```
LOAD_BATCHES_WHEN_MORE_SPACE_IS_AVAILABLE_IN_FIRST_BUFFER = 1
```

__init__(type, plant, schedule) - The class must be initialized with a plant and a schedule. Therefore, the schedule must be created prior to the simulator. An empty list to hold all the executions are also made. The variables 'machineTaskChoosingLogic' and 'introduceNewBatchesLogic' are initialized with zero, and must be set later.

simulationLoop(numberOfWafersTotal, batchSize) - First of all, it finds the number of batches we need with the function above. It then creates a new batch and appends it to the schedules list of batches that have not joined the system yet. To send the first batch into the system the 'scheduleEventBatchEntersFirstBuffer' is called. This function has a while loop, which means it keeps going until all the scheduled events have been executed.

When we are going to simulate 1000 wafers entering the system, with different batchSizes - this function finds the number of batches we need to create, by making new batches until we are *above* numberOfWafersTotal. A while-loop is used to ensure this. This could have been solved in other ways to ensure exactly 1000 wafers were produced. We elaborate on our choice in the section "Explanation and Discussion" below.

simulationHardCodeStart() - This function does the same as the one above, without making batches first. This is because we need this function to run through the schedule events, when we have "hard coded" batches into the first buffer.

executeEvent(event) - Changes the schedule's current date to the event's date. Goes to another execute function depending on the type of the event.

executeEventBatchEntersFirstBuffer(event) - As the name says, this function is called when a batch is finished loading - and enters the first buffer. It alerts the plant that the batch has entered the system, and then alerts the machines that new work can be done. The reason we need a separate function for when the batch is joining the first buffer, is because batches entering the system have no given outputBuffer, since it has not been doing any tasks yet. Therefore we find the buffer it is going to, through the first tasks input buffer.

executeEventBatchEntersBuffer(event) - This function does the exact same thing as the above, but this time the buffer we are loading in is found through the previous tasks output buffer. This function also changes the state of the machine, to show that the machine is now available for new work.

executeEventLoadMachineFromBuffer(event) - This function alerts the plant that the batches are now in the machine. It then schedules a new event - that the machine should start operating on the batch. After the batches are loaded into the machine, the machines (except this one) are looking to see if any new work can be done.

executeEventMachineFinishedOperatingOnBatch(event) - Called when the machine is finished operating on its batch. It then makes a new schedule, that the batch should join the task's output buffer.

machinesLookForWork() - This function is called on each time a machine is finished loading batches and each time a batch is moved to a new buffer. For each machine this function checks if any new tasks can be done. First, it checks if the machine is already operating on a batch. Secondly, it checks if the "machine" is the last final inventory machine. In both of these cases - a task can not be done. Furthermore, it checks if a task can be done, based on the logic described in the machine-class selectNextBatch-function. If a task can be done, a new event is scheduled, that starts the loading of batches into the machine.

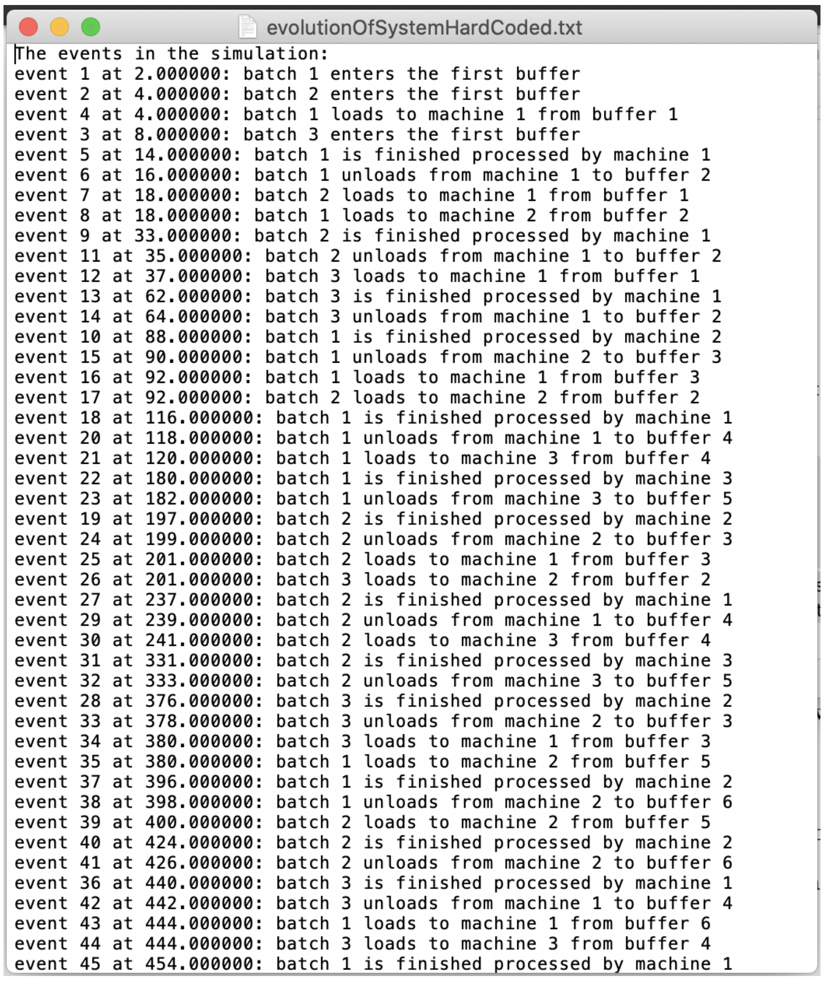
Printer

To print the current state of the simulation, we have chosen to make a printer function that prints each event that is executed. When a simulation is processing, all executions are appended to a list. This list can be used to print out and show how the simulation went. For instance, we hard coded in three batches at different dates into the first buffer:

```
   testBatch20 = testPlant.newBatch(20, 1)
   testBatch30 = testPlant.newBatch(30, 2)
   testBatch50 = testPlant.newBatch(50, 3)
   event1 = schedule.scheduleEvent(Event.BATCH_ENTERS_FIRST_BUFFER, 2, testBatch20,
testPlant.findFirstTask())
```

```
    event2 = schedule.scheduleEvent(Event.BATCH_ENTERS_FIRST_BUFFER, 4, testBatch30,
testPlant.findFirstTask())
    event3 = schedule.scheduleEvent(Event.BATCH_ENTERS_FIRST_BUFFER, 8, testBatch50,
testPlant.findFirstTask())
```

To see how the system evolves, check "evolutionOfSystemHardCoded.txt". A picture of the text-file is shown below.


evolutionOfSystemHardCoded.txt

```
The events in the simulation:
event 1 at 2.000000: batch 1 enters the first buffer
event 2 at 4.000000: batch 2 enters the first buffer
event 4 at 4.000000: batch 1 loads to machine 1 from buffer 1
event 3 at 8.000000: batch 3 enters the first buffer
event 5 at 14.000000: batch 1 is finished processed by machine 1
event 6 at 16.000000: batch 1 unloads from machine 1 to buffer 2
event 7 at 18.000000: batch 2 loads to machine 1 from buffer 1
event 8 at 18.000000: batch 1 loads to machine 2 from buffer 2
event 9 at 33.000000: batch 2 is finished processed by machine 1
event 11 at 35.000000: batch 2 unloads from machine 1 to buffer 2
event 12 at 37.000000: batch 3 loads to machine 1 from buffer 1
event 13 at 62.000000: batch 3 is finished processed by machine 1
event 14 at 64.000000: batch 3 unloads from machine 1 to buffer 2
event 10 at 88.000000: batch 1 is finished processed by machine 2
event 15 at 90.000000: batch 1 unloads from machine 2 to buffer 3
event 16 at 92.000000: batch 1 loads to machine 1 from buffer 3
event 17 at 92.000000: batch 2 loads to machine 2 from buffer 2
event 18 at 116.000000: batch 1 is finished processed by machine 1
event 20 at 118.000000: batch 1 unloads from machine 1 to buffer 4
event 21 at 120.000000: batch 1 loads to machine 3 from buffer 4
event 22 at 180.000000: batch 1 is finished processed by machine 3
event 23 at 182.000000: batch 1 unloads from machine 3 to buffer 5
event 19 at 197.000000: batch 2 is finished processed by machine 2
event 24 at 199.000000: batch 2 unloads from machine 2 to buffer 3
event 25 at 201.000000: batch 2 loads to machine 1 from buffer 3
event 26 at 201.000000: batch 3 loads to machine 2 from buffer 2
event 27 at 237.000000: batch 2 is finished processed by machine 1
event 29 at 239.000000: batch 2 unloads from machine 1 to buffer 4
event 30 at 241.000000: batch 2 loads to machine 3 from buffer 4
event 31 at 331.000000: batch 2 is finished processed by machine 3
event 32 at 333.000000: batch 2 unloads from machine 3 to buffer 5
event 28 at 376.000000: batch 3 is finished processed by machine 2
event 33 at 378.000000: batch 3 unloads from machine 2 to buffer 3
event 34 at 380.000000: batch 3 loads to machine 1 from buffer 3
event 35 at 380.000000: batch 1 loads to machine 2 from buffer 5
event 37 at 396.000000: batch 1 is finished processed by machine 2
event 38 at 398.000000: batch 1 unloads from machine 2 to buffer 6
event 39 at 400.000000: batch 2 loads to machine 2 from buffer 5
event 40 at 424.000000: batch 2 is finished processed by machine 2
event 41 at 426.000000: batch 2 unloads from machine 2 to buffer 6
event 36 at 440.000000: batch 3 is finished processed by machine 1
event 42 at 442.000000: batch 3 unloads from machine 1 to buffer 4
event 43 at 444.000000: batch 1 loads to machine 1 from buffer 6
event 44 at 444.000000: batch 3 loads to machine 3 from buffer 4
event 45 at 454.000000: batch 1 is finished processed by machine 1
```

Implementation

To start a simulation, one could either write hardwritten code to load batches into the first buffer, or start a simulation loop. Before you start a simulation, a schedule and simulator object has to be made. When a batch is going to load into the first buffer, a new event that takes 2 seconds is scheduled. After this event has been exeuted, the machines are alerted and looks for new batches that can be processed. There is now a batch in the first buffer, and the first machine therefore has a job to do. A new event is scheduled, that this batch should be loaded into the machine, this also takes two seconds. Now the machine is occupied with loading this batch inside. When this event is executed and the batch is in the machine, a new event is scheduled, that says the processing should begin. When this event is executed, the machine is finished operating on the batch, and a new event that says that batch should be unloaded is made. When the batch is finished loading into the next buffer, the machines are alerted and again looks for work. This process repeats until the batch has been through all the tasks. The batch is then stored in the final buffer.

Task 3

In this task we implement an optimizer class that changes several different parameters, and finds the best combination that processes all the batches the fastest.

class Optimizer

__init__(plant, numberOfWafersTotal) - The optimizer class must be initialized with a plant and the number of wafers we want to produce. It also makes a dictionary that is later going to store all the simulations. It also makes three lists, that is our parameters:

- Machine task choosing logics
- Batch introduction-logic
- Batch size

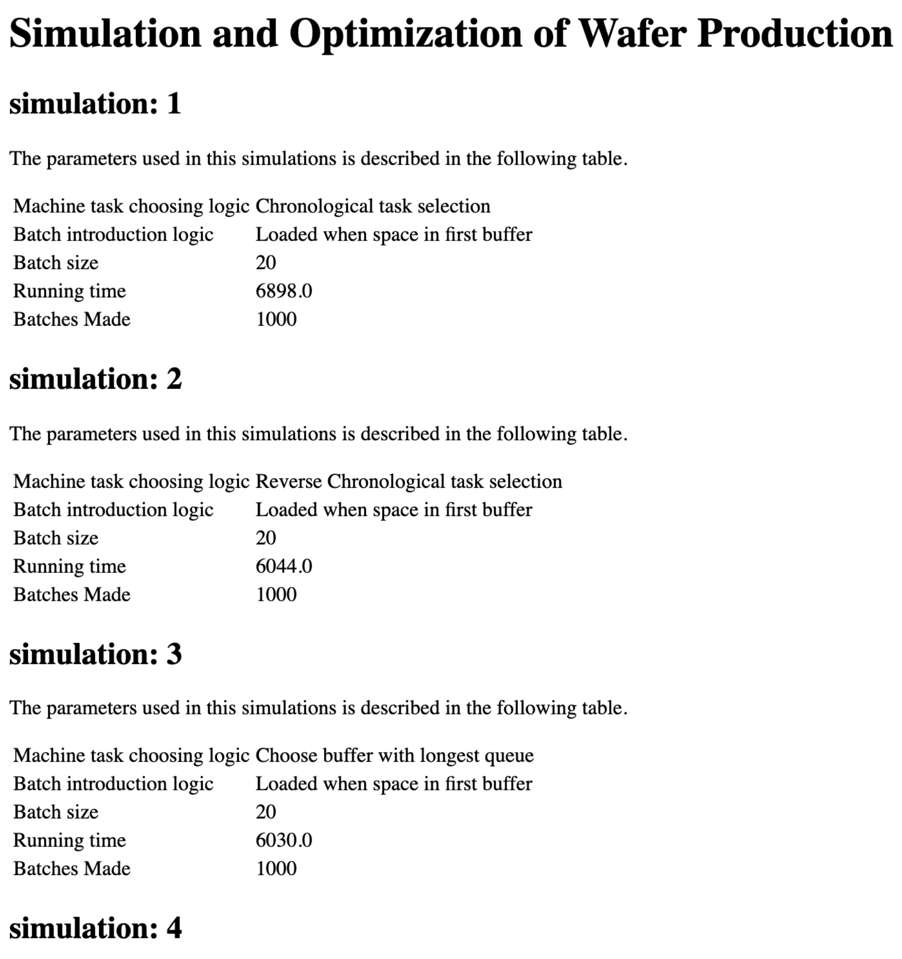
The code currently only has the batches between 20-50 that are divisible by 1000. This can be changed. We have tested several different batch-sizes. Variables that are later going to store the best time and the best simulations are also made.
The number of batch introduction logics we ended up with was, as multiple times already stated, one. For further comments on this, see "Explanation and discussion"-section below.

startOptimization() - Has a simulation number that increases for each simulation. This function iterates through all the different combinations of parameters. It then makes a dictionary, to save the current combination. It then starts the simulation, and saves the time it took. If the time was quicker than the other ones, this particular simulation is saved.

startSimulationAndReturnTimeItTook(machineLogic, batchIntroduceLogic, batchSize) - The function above calls this function each time it wants to start a new simulation with new parameters. This function makes a new schedule and simulator object, sets the machine choosing logic and batch introduction logic and then starts a new simulation loop with these settings.

Printer

We chose to make a new class **HTMLPrinter** to print the stats of all simulations. The HTMLfile can be viewed in the 'report.html'-file. Stats of all simulations are simulations are presented, and the best of them is highlighted at the end of the report, making it easily accessible to the client.



Simulation and Optimization of Wafer Production

simulation: 1

The parameters used in this simulations is described in the following table.

Machine task choosing logic	Chronological task selection
Batch introduction logic	Loaded when space in first buffer
Batch size	20
Running time	6898.0
Batches Made	1000

simulation: 2

The parameters used in this simulations is described in the following table.

Machine task choosing logic	Reverse Chronological task selection
Batch introduction logic	Loaded when space in first buffer
Batch size	20
Running time	6044.0
Batches Made	1000

simulation: 3

The parameters used in this simulations is described in the following table.

Machine task choosing logic	Choose buffer with longest queue
Batch introduction logic	Loaded when space in first buffer
Batch size	20
Running time	6030.0
Batches Made	1000

simulation: 4

Just for fun, and to learn HTML even more , we decided to make a table that well represents the result. Since this is not a part of the task we won't explain in detail the implementation of this table, but we present it below. We made it as a separate file, with the class 'AdditionalHTMLPrinter'. **NB:** The 'report.html' will work for any chosen number of simulations and settings, but the additional printer only works for our exact choices of parameters.

Explanation and discussion

Task 1 - Testing

We test the task as recommended in lectures and assignment texts: The task is tested by hard-coding the plant illustrated in the assignment by step-by-step, starting with machine 1, 2, 3 and the help-machine 4 being the finished goods. Then buffers with capacity constraints are introduced. For machine 4, the capacity is set to 100000000000 (for practical purposes, infinite capacity).

Task 2 - Testing

The task is tested by loading batches at different times, as recommended in the assignment text. The different events are printed to the file evolutionOfSystemHardCoded. This is illustrated on page 11 and 10 in this document. Batches of different sizes at different times are tested. The printing looks good, but of less interest to the client.

We then tested to reset the plants and ran the simulation starting by the simulation loop.

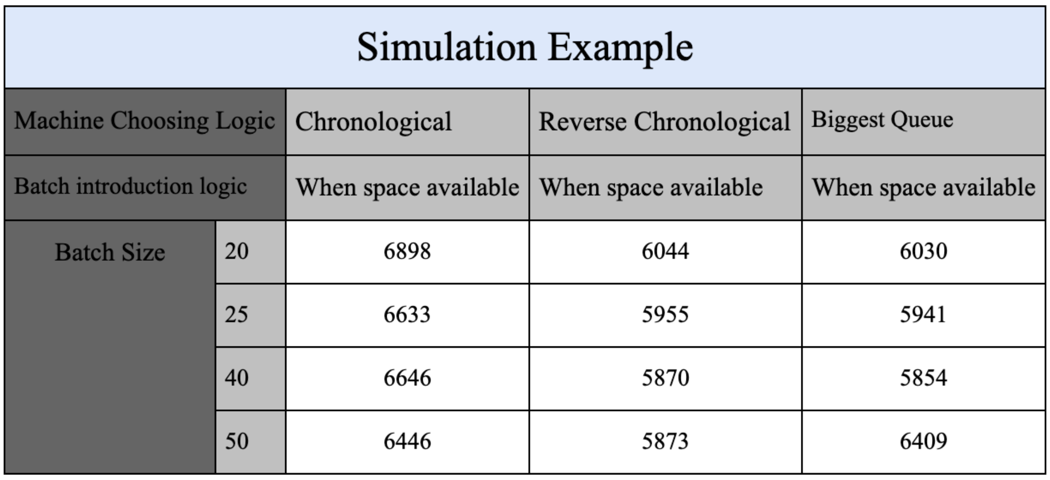
Task 3 - Testing

We tested the optimizer with several different batch sizes, but the current ones are all convenient giving the 1000 wafers to be produced. Below are examples of our extra HTML-table that we found quite pleasant. It highlights the iterative process by presenting the whole simulation matrix:

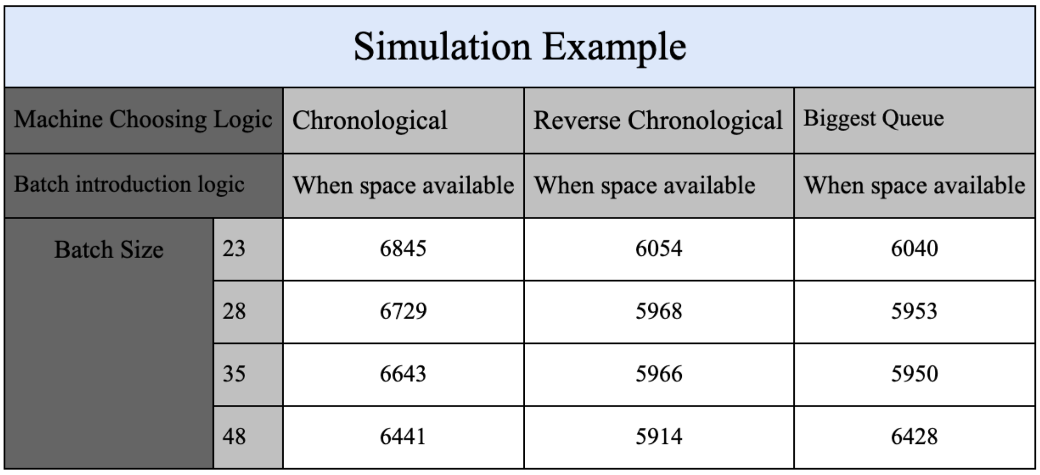
When optimizing the production plan, we need to manually narrow in on the batch range indicated by the first simulation. We will now present an example of how the client should use the program.

Running the first simulation, we get the results in the first table. Clearly the best times in this simulation were between 25 and 40 for the best logics. The problem is not linear, but we assume it to be something close to convex with one optimal solution, hence we zoom the range to 23-48 and run a new simulation. Surprisingly, the new results were not better, and a last check is made by trying around a batch range of 40.

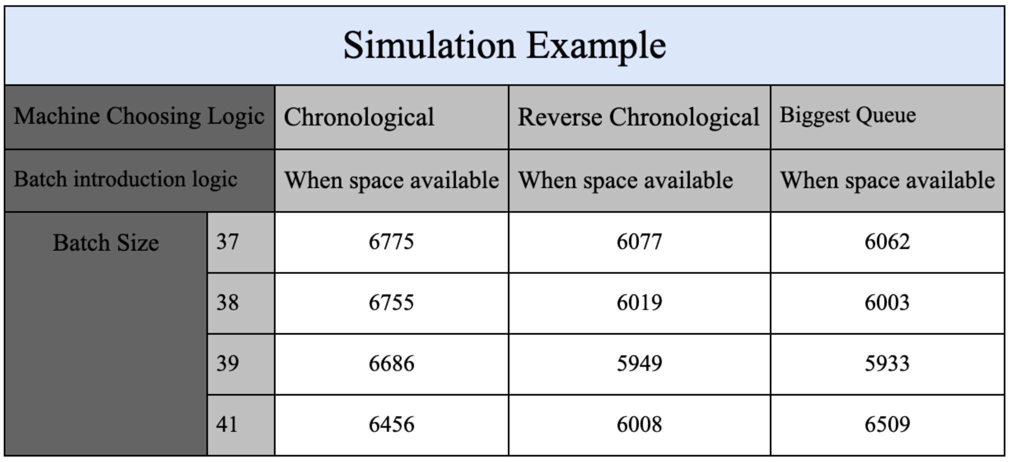
<u>Based on this, we conclude that the best production is done with batch size of 40 with the machines following a biggest queue operation-policy. The time was 5854.</u>



Simulation Example				
Machine Choosing Logic		Chronological	Reverse Chronological	Biggest Queue
Batch introduction logic		When space available	When space available	When space available
Batch Size	20	6898	6044	6030
	25	6633	5955	5941
	40	6646	5870	5854
	50	6446	5873	6409



Simulation Example				
Machine Choosing Logic		Chronological	Reverse Chronological	Biggest Queue
Batch introduction logic		When space available	When space available	When space available
Batch Size	23	6845	6054	6040
	28	6729	5968	5953
	35	6643	5966	5950
	48	6441	5914	6428



Simulation Example				
Machine Choosing Logic		Chronological	Reverse Chronological	Biggest Queue
Batch introduction logic		When space available	When space available	When space available
Batch Size	37	6775	6077	6062
	38	6755	6019	6003
	39	6686	5949	5933
	41	6456	6008	6509

On the choice of batch introduction-logic:

In the simulation, new batches are introduced to the system when there is available space in the first buffer. At first glance, this may seem insufficient when trying to optimize the time to produce x number of wafers at the shortest possible time. But after discussions in exercise hours and considering the vast solution space of the optimization problem, we found it highly reasonable that machine operation policies and a varying batch size would alone create the needed solution space for a great solution. We did not find any other introduction policies being intuitively superior to our current policy. But we actually tried some others:

- These policies were in a sense more restrictive, requireming not only the first input buffer to have space, but also input and output buffers of other tasks to also have available space. This did not better the simulation results, but increased the complexity and running time of simulations significantly.

- We also tried to load new batches into the system when a batch is finished with all its operations. This was very insufficient, compared to the batch introduction-logic we chose.

From optimization courses, it is known that a relaxed optimization problem will *at least* have an optimal solution as good as the more restrictive one, and in most cases better. Although this is not exactly a less-than or greater-than restriction (it is an equality constraint), we believe it restricted our solution space. We may be wrong, but our observations were indicative of this.

As stated earlier, a policy of Chronological-selection combined with always filling up the first buffer, may hamper progress throughout the plant. From a practical point of view, this increases the possibility of high levels of WIP (work-in-progress) goods which we often want to minimize (at least keep low). But when we deviate from the chronological machine policy to the others, this problem becomes negligible (if it even exists).

On the number Wafers produced vs. keeping batch size fixed:

We chose to produce batches until we ensured a production at least equal to the number of Wafers needed. That means, if the one issue is that the number of wafers (here: 1000) may not be divisible by the batch size we want to test. This could be solved by breaching the constraint saying that all batches we introduce need to be the same size, for example by having batch sizes of one size and adjusting the last one to make it 1000 in total. We chose to stay strict on the fixed batch size, due to more practical reasons. We know from other courses and industry experience that standardization in mass production is preferred, and “overproduction” of a few wafers would likely not be a problem. Neither huge inventory costs or deterioration of the wafers are expected because of minor overproduction.