Optimization report

In this report, we look at how to improve a factory process that makes semiconductor wafers. Our goal is to make 1000 wafers as quickly as we can. Check out the documentation for the python simulator to understand how it works and the rules it must follow.

We have investigated three main parameters to improve the production performance:

A. The heuristic used by each unit to select the next batch to process.
B. The way the 1000 wafers are grouped into batches.
C. The times at which the batches are loaded into the input buffer of the production line.

Our approach for examining the three parameters involved holding all but one constant, and then observing how performance shifted as the variable changed. In this report, we will present the experiments conducted for each parameter, detailing our attempts and findings. Ultimately, we will share our conclusions regarding the optimal parameter settings we discovered.

We started with trying to find the best task prioritization for the units. Then we tried to find the best way to group wafers into batches. Lastly, we experimented with the times at which the batches were loaded into the first task.

Parameter A: The heuristic used by each unit to select the next batch to process

These are the five heuristic methods we tested:

1. We prioritized tasks in the order they appear in the production line, giving higher priority to earlier tasks.
2. We prioritized tasks based on their processing time, giving higher priority to tasks with longer processing times.
3. We prioritized tasks based on their processing time, giving higher priority to tasks with shorter processing times.
4. We used a dynamic approach, prioritizing tasks based on how full their input buffer was, always choosing the least full. We now this is weird, but we tried the fullest one as well, but it was bad.
5. Our last approach where to test all the different combinations of the task orders to check which one performed the best.

Here is the result for the different heuristics tested. The settings are 1000 wafers split in 20 batches with 20 wafers each.

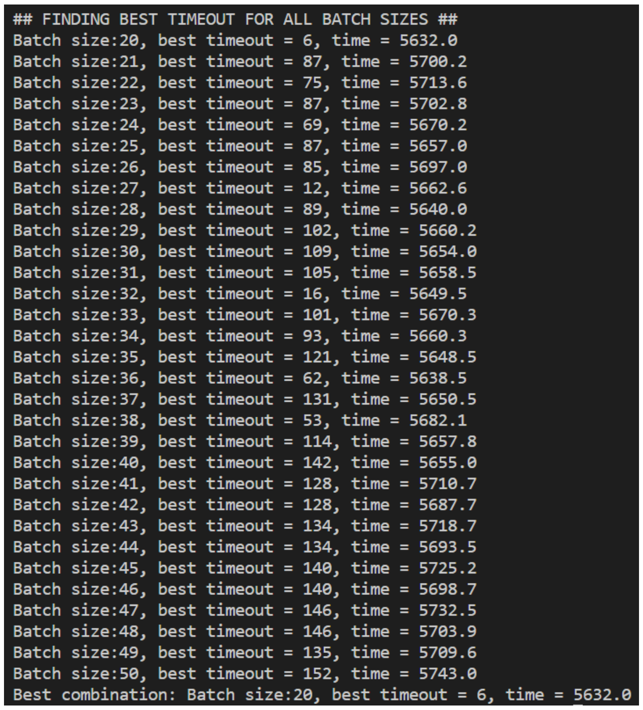
Heuristic method	Task prioritization	Time used
1	[[1, 3, 6, 9], [2, 5, 7], [4, 8]]	6583.0
2	[[3, 6, 1, 9], [2, 5, 7], [4, 8]]	6583.0
3	[[9, 1, 6, 3], [5, 7, 2], [8, 4]]	5734.0
4	Dynamically changing	5633.0
5	[[1, 3, 6, 9], [5, 7, 2], [4, 8]]	5652.0

We concluded that after we tested all the combinations with the settings described over that dynamically changing was the best task prioritization.

Parameter B: The times at which the batches are loaded into the input buffer of the production line

This is the methods we used for timing when the batches are loaded into the start buffer of the production line.

1. We have incorporated infinite capacity into the start buffer to facilitate the immediate loading of batches as space becomes available. Consequently, there is no need for an additional buffer to schedule data loading into the start buffer, as this would only introduce unnecessary complexity without impacting the outcome.
2. We added functionality for scheduling so we can add a timeout between when batches is added to the start buffer. This required us to create a additional action for our event system. If the start buffer is full and the batch cannot be added with the timeout we scheduled a new event will be added to the event queue a timeout into the future. We then did an experiment to see what the best timeout was for all the different ways of dividing 1000 wafers into equal batch sizes. We did this experiment with the dynamically changing task prioritization. We found that the best timeout was 6 for batch size 20.

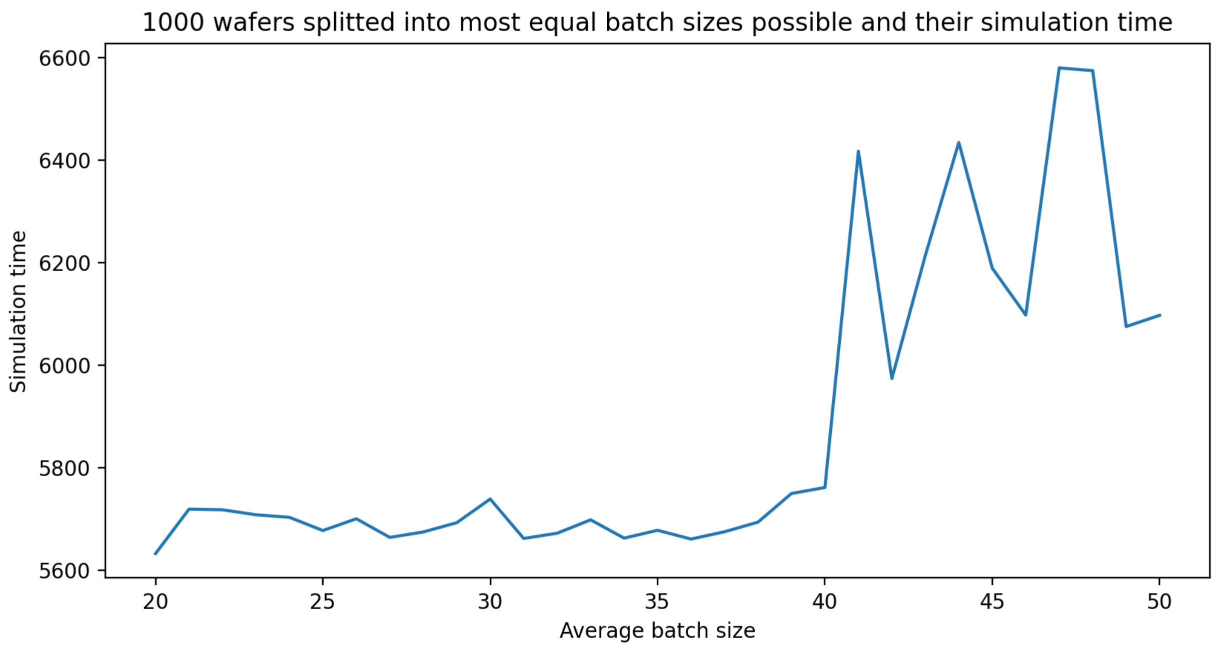


```
## FINDING BEST TIMEOUT FOR ALL BATCH SIZES ##
Batch size:20, best timeout = 6, time = 5632.0
Batch size:21, best timeout = 87, time = 5700.2
Batch size:22, best timeout = 75, time = 5713.6
Batch size:23, best timeout = 87, time = 5702.8
Batch size:24, best timeout = 69, time = 5670.2
Batch size:25, best timeout = 87, time = 5657.0
Batch size:26, best timeout = 85, time = 5697.0
Batch size:27, best timeout = 12, time = 5662.6
Batch size:28, best timeout = 89, time = 5640.0
Batch size:29, best timeout = 102, time = 5660.2
Batch size:30, best timeout = 109, time = 5654.0
Batch size:31, best timeout = 105, time = 5658.5
Batch size:32, best timeout = 16, time = 5649.5
Batch size:33, best timeout = 101, time = 5670.3
Batch size:34, best timeout = 93, time = 5660.3
Batch size:35, best timeout = 121, time = 5648.5
Batch size:36, best timeout = 62, time = 5638.5
Batch size:37, best timeout = 131, time = 5650.5
Batch size:38, best timeout = 53, time = 5682.1
Batch size:39, best timeout = 114, time = 5657.8
Batch size:40, best timeout = 142, time = 5655.0
Batch size:41, best timeout = 128, time = 5710.7
Batch size:42, best timeout = 128, time = 5687.7
Batch size:43, best timeout = 134, time = 5718.7
Batch size:44, best timeout = 134, time = 5693.5
Batch size:45, best timeout = 140, time = 5725.2
Batch size:46, best timeout = 140, time = 5698.7
Batch size:47, best timeout = 146, time = 5732.5
Batch size:48, best timeout = 146, time = 5703.9
Batch size:49, best timeout = 135, time = 5709.6
Batch size:50, best timeout = 152, time = 5743.0
Best combination: Batch size:20, best timeout = 6, time = 5632.0
```

Parameter C: The way the 1000 wafers are grouped into batches

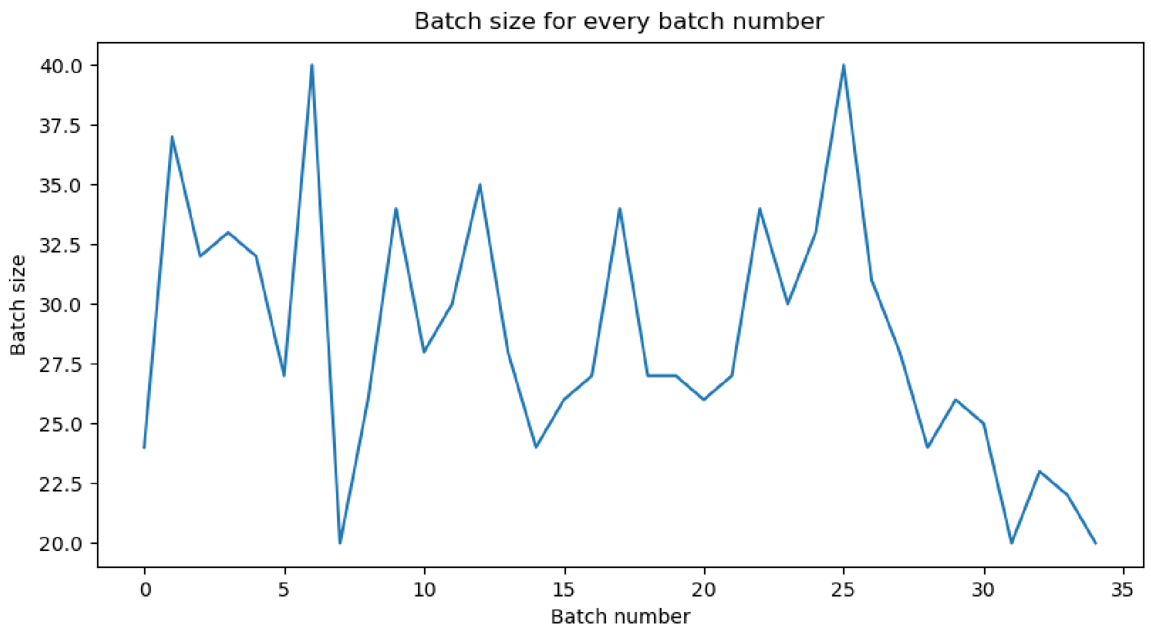
These are the three methods we tested for finding the best way of grouping 1000 wafers into batches. The task prioritization we used was the dynamic method.

1. Our first method was to divide the 1000 wafers into equal batches. If 1000 isn't divisible by the batch size, we create an extra batch for the remaining wafers if it is over or equal to 20 left, or spread the extra wafers among the batches if it is less than 20 left. The graph below shows the different time used on the different batch sizes. With this experiment we concluded with that the best batch size if every batch was going to be the same was 20. The simulation time was 5633.0 for this batch size.



2. Our second method was pure brute force. We developed a function to generate random batch sizes totaling 1000 and applied it in our iterative brute-force function, saving the initial batches with the best outcome in a CSV file. After roughly 10k iterations, we achieved a simulation time of 5630.0. Despite recognizing the vast number of potential batch combinations, we proceeded with this method to test our chances.
3. For our third method we created a genetic algorithm. Our staring population was a set of random initial batches and the best initial batches we had saved. We always passed the top 10% to the next generation. We also mutated a copy of the top 10% that we passed on to the next generation. We created new random initial batches for the 80% that didn't make it to the next generation. We constantly improved our algorithm and kept our best results. We managed to get the simulation time down to 5589.8 in 1000 generations using this method.

We also examined the batch sizes of our top initial batches, seeking a discernible pattern through visual analysis. However, we could not find any pattern with the human eye. Here is a graph displaying the batch sizes for our best initial batches:



Final result:

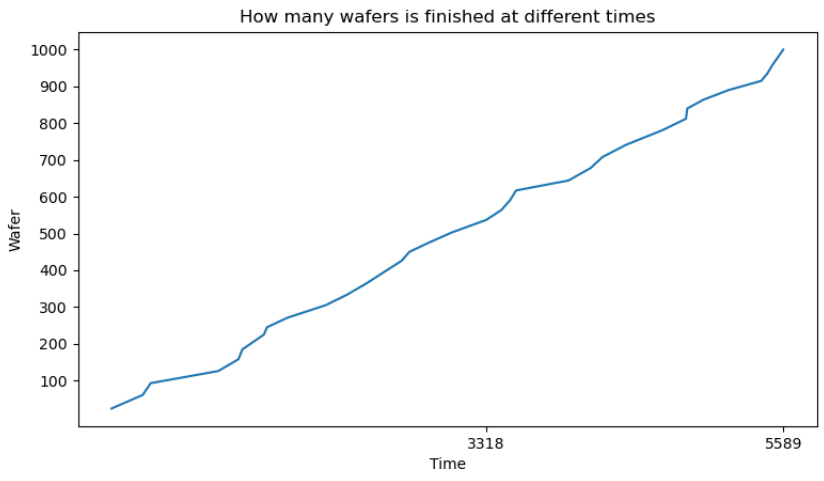
Task prioritization: dynamically changing and prioritizes the task with the least wafers in its input buffer.

Timeout between inserting batches: 6

Initial batches:
[24,37,32,33,32,27,40,20,26,34,28,30,35,28,24,26,27,34,27,27,26,27,34,30,33,40,31,28,24,26,25,20,23,22,20]

Time: **5589.8**

Here is a graph showing how many wafers is finished at the different timestamps:


How many wafers is finished at different times
Wafer
Time