Micron NUS-ISE Business Analytics Case Competition 2025

BIZy

Contents

Question 1

Part a)

First, we define all the constraints required to formulate the problem. Let $x_{q,i}$ be the number of wafers loaded per week for Node i in quarter q, and let $y_{q,i}$ be the yield of Node i in quarter q. Note that for all q, $y_{q,1} = 0.98$.

1. **TAM constraint**: Given the formula for the total GB output, which should meet demand per quarter

$$GB_{q,out} = 13 \cdot (x_{q,1} \cdot 100000 \cdot 0.98 + x_{q,2} \cdot 150000 \cdot y_{q,2} + x_{q,3} \cdot 270000 \cdot y_{q,3})$$

   for each quarter q, one must have

$$|GB_{q,out} - TAM_q| \leq 2000000000$$

2. **Load change constraint**: The load in each quarter must satisfy the condition

$$|x_{q,n} - x_{q-1,n}| \leq 2500$$

   for each node n and quarter q.

3. **Load non-negativity constraint**: The load in each quarter must satisfy $x_{q,n} \geq 0$.

4. **Initial conditions**: The initial conditions are $x_{1,1} = 12000, x_{2,1} = 5000,$ and $x_{3,1} = 1000$.

Our objective is to maximize the total contribution margin to maximize profit. Mathematically, optimize

$$\max \left(\sum_q GB_{q,out} * 0.002 \right)$$

while meeting the constraints given to us. Here is the code snippet to solve the linear optimization.

Python Code

```
import pandas as pd
import numpy as np
from pulp import LpMaximize, LpProblem, LpVariable, lpSum, LpStatus

# Store results
sequential_results = {}

# Sequentially solve for each quarter
for i, q in enumerate(quarters):
    # Create a new optimization problem for each quarter
    model = LpProblem(f"Maximize_Contribution_{q}", LpMaximize)

    # Decision variables for current quarter
    x = {n: LpVariable(f"x_{q}_{n}", lowBound=0) for n in nodes}

    # Objective: Maximize contribution for this quarter
    model+=lpSum(0.002*13*x[n]*ypw[n][q] for n in nodes), "Quarter_Contribution"

    # TAM constraints for this quarter
    model+=13*lpSum(x[n]*ypw[n][q] for n in nodes)-TAM_data[q]<=2*1e9
    , f"TAM_upper_bound_{q}"
    model+=TAM_data[q]-13*lpSum(x[n]* ypw[n][q] for n in nodes)<=2*1e9
    , f"TAM_lower_bound_{q}"
    print(sequential_results)
    # Loading change constraints
```

```
    if i > 0:
        prev_q = quarters[i - 1]
        for n in nodes:
            prev_loading = sequential_results[prev_q][n]
            model += x[n] - prev_loading <= 2500, f"Loading_change_up_{q}_{n}"
            model += prev_loading - x[n] <= 2500, f"Loading_change_down_{q}_{n}"
    else:
        for n in nodes:
            model += x[n] == initial_loading[n], f"Initial_loading_{n}"

    # Solve the model
    model.solve()

    # Store results
    quarter_solution = {n: math.floor(x[n].varValue) for n in nodes}
    sequential_results[q] = quarter_solution
```

Part b)

For this part, let W be the set of workstations, and denote by w for each workstation in W. Moreover, denote Loading$_q$ for the loading in quarter q, Utilization$_w$ for utilization in workstation w, and MinuteLoad$_{w,i}$ for the minute load in workstation w for Node i.

Part b)(i)

Denote by T_q the tool requirement for quarter q, and $T_{q,w}$ the tool requirement for workstation w in quarter q. Then we can obtain T_q as follows.

$$T_q = \sum_{w \in W} T_{q,w}$$

where each $T_{q,w}$ is obtained as such

$$T_{q,w} = \sum_{i=1}^{3} \left(\frac{\text{Loading}_q \cdot \text{MinuteLoad}_{w,i}}{7 \cdot 24 \cdot 60 \cdot \text{Utilization}_w} \right)$$

and this gives us the desired result, as shown in our table in Excel. We use the code snippet below to aid us with our computations.

Python Code

```
import math
# Initialize an empty dctionary for total tool requirements
tool_requirements = [] # Tool requirement for each workstation
quarter_tool_requirement = [] # Tool requirement for each quarter
workstation_quarter_tool = [] * 8

# Collect the initial tool count for each workstation for Q1'26
for idx, row in workstation.iterrows():
    tool_requirements.append(row["Initial tool count for Q1'26"])



prev_tool = tool_requirements[:]
quarter_tool_requirement.append(sum(tool_requirements))
quarters_new = quarters[:]
del quarters_new[0]
```

```
for q_idx, q in enumerate(quarters_new):
  work_station = []
  for idx, row in workstation.iterrows():
    total_tool_requirement = 0 # This is the total tool requirement for
    # each workstation
    for n_idx, n in enumerate(nodes): # Iterate over each node
      minute_load = row[f"Node {n_idx+1} Minute Load"]
      utilization = row["Utilization (percent)"] / 100
      loading = int(sequential_results[q][n])
      node_tool_requirement = (loading * minute_load) / (7 * 24 * 60 * utilization)
      total_tool_requirement += math.ceil(node_tool_requirement)
    tool_requirements[idx] = math.ceil(total_tool_requirement) # The new initial
    # tool count for the quarter is the total of the prev
    work_station.append(math.ceil(total_tool_requirement))
  workstation_quarter_tool.append(work_station)
  quarter_tool_requirement.append(sum(tool_requirements))

print(quarter_tool_requirement)
print(workstation_quarter_tool)
```

Part b)(ii)

Denote by $C_{w,q}$ the additional CAPEX incurred in workstation w at quarter q, and denote by CAPEX$_w$ the CAPEX per tool for workstation w. Let us define the tool requirements of the previous quarters before quarter q as

$$TR_q = \{T_{w,k} \ : \ k < q\}$$

Using this formula, perform this calculation for each $q \geq 2$.

$$C_{w,q} = \max(T_{w,q} - \max\{TR_q\}, 0) \times \text{CAPEX}_w$$

where $\max\{TR_q\}$ is the maximum element of TR_q. The rationale behind this formula is that if the previous quarter's tool requirement is greater than the current quarter tool's requirement, then we take it to be $\$0$ of additional CAPEX incurred, since there is no selling of tools. Using this formula would then give us our results in the table in Excel.

Part b)(iii)

The net profit for each quarter q is therefore

$$\text{Profit}_q = GB_{q,out} \times 0.002 - \sum_{w \in W} C_{w,q}$$

We then sum the profit for each quarter, as shown below

$$\sum_{q=1}^{8} \text{Profit}_q$$

to obtain the total net profit for the 8 quarters. The result can be seen in our table in Excel.

Part b)(iv)

In order to maximize the total profit across 8 quarters, rather than only maximizing gain from total GB output, we need to minimize the number of additional tools to take into account of the production of the wafers.

Python Code

```
model = LpProblem("Maximize_Total_Profit", LpMaximize)

# Decision Variables:
x = {(q,n):LpVariable(f"x_{q}_{n}", lowBound=0) for q in quarters for n in nodes}

# delta_y[w,q]: additional tools purchased at workstation w in quarter q (integer)
delta_y={(w,q):LpVariable(f"delta_{w}_{q}", lowBound=0,cat="Integer")
for w in ws_names
for q in quarters}

# y[w,q]: cumulative number of tools available at workstation w in quarter q (integer)
y={(w,q):LpVariable(f"y_{w}_{q}",lowBound=0,cat="Integer")
for w in ws_names for q in quarters}

model += lpSum(
    contrib_margin[q] * lpSum(x[q, node] * yield_per_wafer[node][q] for node in nodes)
    -
    lpSum(capex[w] * delta_y[w, q]
    for w in ws_names)
    for q in quarters
), "Total_Profit"

# 1. Production (TAM) Constraint for each quarter:
for q in quarters:
    model+=lpSum(13*x[q,n]*yield_per_wafer[n][q] for n in nodes)
    <=(TAM_target[q]+2)*1e9, f"TAM_upper_{q}"
    model+=lpSum(13*x[q, n]* yield_per_wafer[n][q] for n in nodes)
    >= (TAM_target[q]-2)*1e9, f"TAM_lower_{q}"

# 2. Workstation Capacity Constraints for each workstation and quarter:
for q in quarters:
    for w in ws_names:
        model+=lpSum(minute_load[w][n]*x[q,n] for n in nodes)
        <=y[w,q]*10080*(utilization[w]/100), f"Capacity_{w}_{q}"

# 3. Cumulative Capacity Constraints:
first_q = quarters[0]
for w in ws_names:
    model += y[w, first_q] == init_tools[w] + delta_y[w, first_q],
    f"InitCapacity_{w}_{first_q}"

for idx, q in enumerate(quarters[1:], start=1):
    prev_q = quarters[idx - 1]
    for w in ws_names:
        model += y[w, q] == y[w, prev_q] + delta_y[w, q],
        f"CumulCapacity_{w}_{q}"

# 4. Loading Constraints:
# Initial loading values for wafer production (number of wafers) for Q1'26
init_loading = {"Node 1": 12000, "Node 2": 5000, "Node 3": 1000}
first_q = quarters[0]
for node in nodes:
    model += x[first_q, node] == init_loading[node], f"Init_loading_{node}"
```

```
# For subsequent quarters, the change in production for each node is
# restricted to ±2500 wafers.
for idx, q in enumerate(quarters[1:], start=1):
    prev_q = quarters[idx - 1]
    for node in nodes:
        model += x[q, node] - x[prev_q, node] <= 2500
        , f"Loading_change_pos_{node}_{q}"
        model += x[prev_q, node] - x[q, node] <= 2500
        , f"Loading_change_neg_{node}_{q}"

model.solve()
```

Here, we added more constraints to quantify that we wanted to minimize the expenditure on tools and how buying tool may change the possible loading to still maximize GB output.

Note that the new loading plan is filled in our table in Excel.

Part c)

Our objective is to maximize total profit. It balances two components: revenues (calculated based on the contribution margin per GB and the total GB output) and costs (including the CAPEX costs of purchasing additional tools). In addition to constraints from part (a), we also add workstation capacity constraints to avoid overloading of workstations. The additional constraint is as follows

$$\text{Loading}_i \cdot \text{MinuteLoad}_{w,i} \leq 7 \cdot 24 \cdot 60 \cdot \text{Utilization}_w$$

where Loading_i denotes loading for Node i, Utilization_w denotes utilization for Workshop w, and $\text{MinuteLoad}_{w,i}$ denotes the minute load for Workshop w for Node i. In addition to preventing overworking of workstations, this also helps to minimize our costs. While this may reduce the profit by $\$11$M, the risk that the workstations overloaded get decreased significantly, greatly reducing chances of not meeting the designed loading plan.

Question 2

Part a)

Part a)(i)

Denote RPT_i to be the RPT for Node i, then using `numpy` and `pandas` in `Python`, it was found that $\mu_{RPT_1} = 2.131$, $\mu_{RPT_2} = 5.992$, and $\mu_{RPT_3} = 3.024$. Further, we also found that $\sigma_{RPT_1} = 0.5$, $\sigma_{RPT_2} = 3.291$ and $\sigma_{RPT_3} = 2.054$, all correct to 3 significant figures. The code used is shown below.

Python Code

```
import numpy as np
import pandas as pd

df = pd.read_csv('RPT.csv')
means = []
means.append(np.mean(df['H_1']))
means.append(np.mean(df['I_2']))
means.append(np.mean(df['J_3']))
print(means)
sd = []
sd.append(math.sqrt(np.var(df['H_1'])))
sd.append(math.sqrt(np.var(df['I_2'])))
sd.append(math.sqrt(np.var(df['J_3'])))
print(sd)
```

Part a)(ii)

Note that if X is a random variable, one has $E(X) = \mu$, where μ is the population mean of X. Since the distribution of RPT_i is random by assumption, and hence it is a random variable, we conclude that for each RPT_i, we have $E(RPT_i) = \mu_{RPT_i}$. Then we have $E(RPT_1) = 2.131, E(RPT_2) = 5.992$, and $E(RPT_3) = 3.024$, all correct to 3 significant figures.

Part a)(iii)

Define TR to be the tool requirement. The expectation value of TR is

$$E(TR) = E\left(\sum_{i=1}^{3} \frac{1}{10080} \cdot \frac{Load_i \cdot RPT_i}{Utilization_i}\right) = \sum_{i=1}^{3} \frac{1}{10080} \cdot \frac{Load_i}{Utilization_i} \mu_{RPT,i} = 7.45$$

correct to 3 significant figures.

Part a)(iv)

We can estimate wafer processing times using Quasi-Monte Carlo (Sobol) sampling. As long as each population follows an empirical distribution, we can use the same methodology with appropriate adjustments, such as fitting a distribution to the dataset and incorporating different constraints into the simulation.

Python Code

```
import matplotlib.pyplot as plt
from scipy.stats import qmc

n_samples = 8192

# Define a function to compute empirical quantile using interpolation
def empirical_quantile(data, u):
    data_sorted = np.sort(data)
    n = len(data_sorted)
    # Create an evenly spaced grid between 0 and 1 for quantiles
    grid = np.linspace(0, 1, n)
    return np.interp(u, grid, data_sorted)

# Function to perform QMC simulation for empirical distribution
def qmc_empirical_sum(data, count, n_samples, block_size=100):
    total_samples = np.zeros(n_samples)
    remaining = count
    # Initialize a Sobol engine with dimension equal to block_size
    while remaining > 0:
        current_block = min(block_size, remaining)
        sobol = qmc.Sobol(d=current_block, scramble=True)
        # Generate quasi random numbers in shape (n_samples, current_block)
        U = sobol.random(n_samples)
        # Map these using the empirical quantile function for each column
        # Loop over current block dimension
        block_vals = np.zeros((n_samples, current_block))
        for j in range(current_block):
            block_vals[:, j] = empirical_quantile(data, U[:, j])
        # Sum along the block dimension and add to total
        total_samples += block_vals.sum(axis=1)
        remaining -= current_block
    return total_samples
```

```
# Use the function for each node and each week's wafer count
qmc_simulation = {}
for node, weekly_counts in wafer_counts.items():
    data = df[node].dropna().values  # in case there are NaNs
    qmc_simulation[node] = []  # Store results per week
    for week_idx, count in enumerate(weekly_counts):
        qmc_total = qmc_empirical_sum(data, count, n_samples)
        qmc_simulation[node].append(qmc_total)

# Plot histograms for each node's QMC simulation
plt.figure(figsize=(10, 6))
for node in wafer_counts.keys():
    for week_idx, week_data in enumerate(qmc_simulation[node]):
        plt.hist(week_data, bins=50, alpha=0.5, label=f'{node} Quarter {week_idx+1}')
plt.title('Histogram of Total Processing Times (QMC Empirical Simulation)')
plt.xlabel('Total Processing Time')
plt.ylabel('Frequency')
plt.legend()
plt.show()

print('QMC Empirical simulation complete')
```

Part a)(v)

From (iv), it was found that total available tool times for workstations H, I, J are $25704, 30240, 6048$ minutes respectively. Denote by TPT_w the total processing time for workstation w. The probability that we are not able to meet our designed output for the week is the probability that at least one of the workstations has total processing time greater than its available tool time. Using this code, we found the probability that the workstations has the total processing time greater than its available tool time

Python Code

```
import numpy as np
list_h1 = [3] * 8
list_i2 = [4,6,8,9,9,9,9,9]
list_j3 = [1,3,3,3,3,3,3,3]
# Define available tool times for each node
available = {'H_1': [x * 8568 for x in list_h1],
'I_2': [x * 7560 for x in list_i2],
'J_3': [x * 6048 for x in list_j3]}

# Calculate probability that
total processing time > available tool time for each node
and each quarter
probabilities_per_quarter = {}

for node in available.keys():
    probabilities_per_quarter[node] = []
    for quarter_idx in range(len(available[node])):
        sim_data = qmc_simulation[node][quarter_idx]  # Get simulated data
        prob = np.mean(sim_data > available[node][quarter_idx])  # Compute probability
        probabilities_per_quarter[node].append(prob)
        print(f'Probability that total time exceeds tool
        time for {node}, Quarter {quarter_idx+1}:', prob)
```

Now that we found $P(TPT_H > 25704), P(TPT_I > 30240), P(TPT_J > 6048)$, we can get the desired probability as follows

$$P(\text{fail}) = 1 - (P(TPT_H \leq 25704) \times P(TPT_I \leq 30240) \times P(TPT_J \leq 6048)) = 0.120$$

correct to 3 significant figures.

Part a)(vi)

Let $P(\text{fail})$ for each week be defined as in (v). Let X be the number of weeks failed, then we have $X \sim Bin(13, 0.113)$. For Quarter 1, we can get the desired probability as follows

$$\begin{aligned} P(X \geq 1) &= P(X > 0) \\ &= 1 - P(X \leq 0) \\ &= 1 - (1 - P(\text{fail}))^{13} \\ &= 1 - (1 - 0.120)^{13} \\ &= 0.809 \end{aligned}$$

Let $P(X \geq 1)_q$ be the probability of a failure in at least 1 week for Quarter q. Using this snippet of code, we can continue to get the desired probabilities for each quarter.

Python Code

```
probabilities_per_quarter = {}

for node in available.keys():
    probabilities_per_quarter[node] = []
    for quarter_idx in range(len(available[node])):
        sim_data = qmc_simulation[node][quarter_idx]  # Get simulated data
        prob = np.mean(sim_data > available[node][quarter_idx])  # Compute probability
        probabilities_per_quarter[node].append(prob)

# Compute overall failure probability per quarter
overall_probabilities_per_quarter = []

for quarter_idx in range(len(available['H_1'])):
    meet_probs = [1 - probabilities_per_quarter[node][quarter_idx]
    for node in available.keys()]
    overall_failure_prob = 1 - (np.prod(meet_probs))**13
    overall_probabilities_per_quarter.append(overall_failure_prob)

print('Done computing probabilities per quarter.')
```

Note that the probabilities for each quarter have been added in our Excel Sheet.

Part a)(vii)

Denote by $P(\text{fail})_i$ the probability of not meeting the designed output for Quarter i. Note that for all Quarters i, we have listed $P(\text{fail})_i$ in our Excel Sheet. Then we can get the probability of success for Quarter i as such

$$P(\text{Success})_i = 1 - P(\text{fail})_i$$

Then the probability of at least 1 quarter failing can be calculated as follows

$$\begin{aligned} P(\text{at least 1 Quarter fails}) &= 1 - P(\text{0 Quarters fail}) \\ &= 1 - \prod_{i=1}^{8} P(\text{Success})_i \\ &= 1 \end{aligned}$$

using the following code

Part b)

Let us depict the distributions of H_1, I_2, and J_3. Note that I_2 and J_3 have bimodal distributions. If we use the

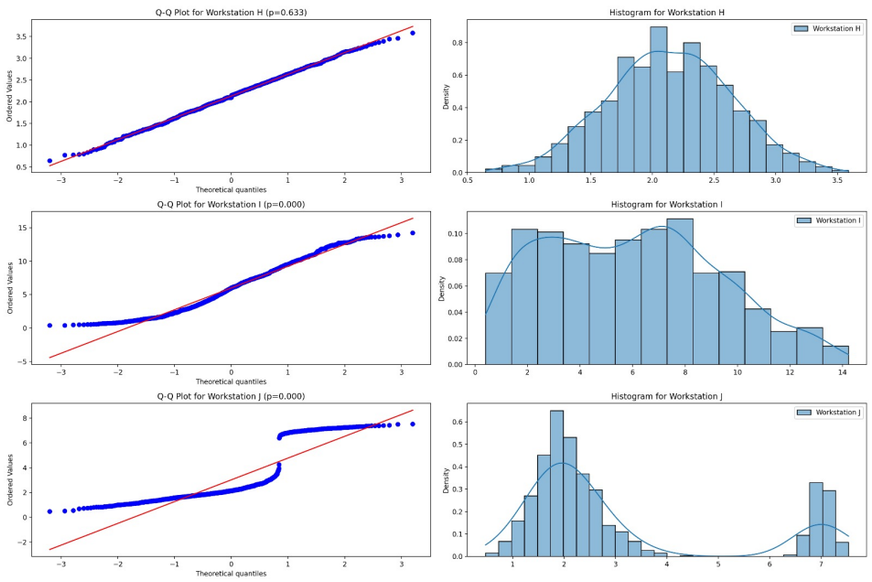

Figure 1: Distribution Depiction of Workstations

median to estimate RPT for J_3, it will be closer to the dominant curve (left). In either case, the median will not be an accurate representation of where the mode lies in the dataset, so it will not be a good estimator of the RPT.

Inspection of the QQ plots reveals distinct distribution characteristics. The distribution of H_1 is approximately Gaussian, whereas I_2 is Gaussian-like but exhibits left and right outliers, and J_3 deviates substantially from Gaussian behavior, showing bimodality. Using the median to estimate the RPT J_3 would bias the estimate toward the dominant (left) mode, failing to represent the true central tendency of the distribution. In contrast, the weighted expected mean is proposed as the estimator. This approach effectively integrates the influence of both modes in bimodal distributions, such as I_2 , rather than collapsing to a single central value. Furthermore, for J_3 , the weighted expected mean better captures the distribution's deviations and complexities. Thus, by simulating the stochastic process by Quasi-Monte Carlo methods to preserve sampling variance, the weighted expected mean showed higher chance to fulfill the design output requirements compared to the median estimator.

Part c)

From the new insights in Q2a) and b), we have decided that we will modify our loading plan by buying 1 extra tool for Workstations I and J respectively. Clearly, the projected profits will be reduced as an extra $\$11$M will be spent for buying the tools, deducted from the profits. However, this can be seen as a long-term investment for the company, as this tool will be used for business operations in the long run, gaining them profit.

Furthermore, we will have more available tool time and this significantly reduces the chances of not meeting the designed loading for each quarter. By buying the tools, it can be seen that the risk level of not meeting the designed loading immediately drops from nearly 100% to 0%.

Quarter	Before	After
Q1/26	0.797773934	0.770627070
Q2/26	0.0	0.0
Q3/26	0.0	0.0
Q4/26	0.999823412	0.0
Q1/27	0.999997357	0.0
Q2/27	0.999998291	0.0
Q3/27	0.999999832	0.0
Q4/27	0.999999863	0.0

The table shows the comparison of probability of not meeting the design output before and after buying an extra tool for workstation I and J. Note that the for Q1/26 is not determined by us, and hence the probability of 0.77 in Quarter 1.

From the distributions of Workstations H, I, J, it can be seen that the median is not an accurate measure due to variability in RPT. To counter this, the extra tools gives us an abundant amount of available tool time to deal with the cases where RPTs are extremely high, or are outliers. Based on this, we would change our answer in Q1 by buying 1 extra tool for Workstations I and J respectively.