Performance Evaluation of Manufacturing Cells

Jinxiang Ma, Ziying Wang, Kirstie Chen

June 23, 2024

1 Introduction

Facing the complexities of a declining economic situation and heightened competition, our organization is tasked with making pivotal decisions regarding the future of our flexible manufacturing cells. These cells have historically been integral to the success of our personal computing division. However, due to strategic directives focused on cost containment and operational efficiency, we are compelled to reevaluate their viability. In response, we have undertaken a comprehensive Data Envelopment Analysis (DEA) to furnish a data-driven recommendation for the potential closure of underperforming cells and to enhance the efficiency of continuing operations, all while navigating the multi-faceted nature of resource utilization and performance criteria across our cells.

2 Data Envelopment Analysis

For our analysis, we define the following input and output variables:

Input:

- Total Cost (TC): This represents the overall expenses incurred by the manufacturing system, measured in millions of dollars. It includes all costs associated with production such as materials, labor, and overhead.
- Work in Process (WIP): This is the number of units that are in the production process but not yet completed. It's a measure of the production system's internal efficiency and is counted in units.
- Employees (EMP): The number of persons employed in the manufacturing system. It's an indication of the labor force involved in the production process.
- Space Requirements (SR): The amount of physical space required for the manufacturing system's operations, measured in thousands of square feet. This could include the factory floor, storage areas, and any additional space dedicated to production.

Output:

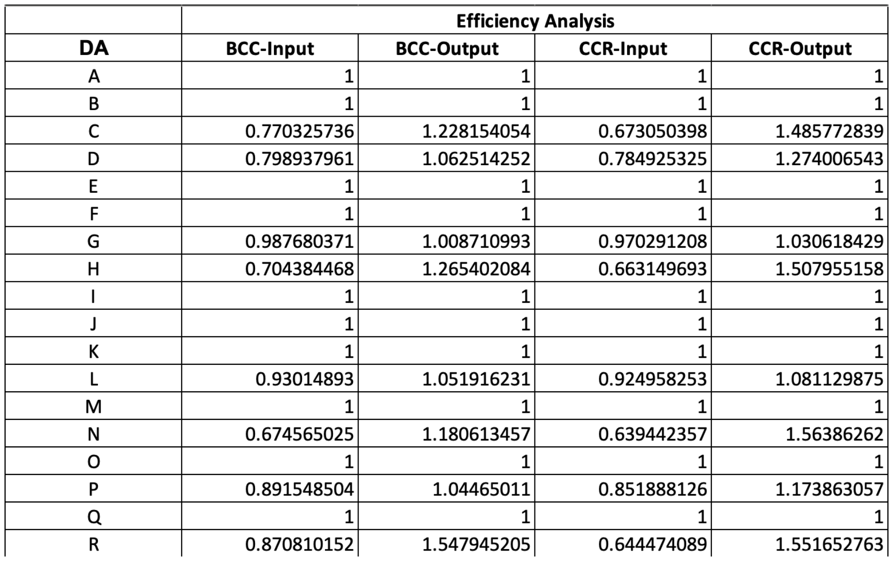
- Throughput (TT): This measures the production system's output rate, quantified in terms of hours per unit of production. It indicates how quickly the system can produce a unit of product.
- Volume Finished (VF): The total number of units produced by the system, given in thousands. This is a direct measure of the system's productivity.
- Product Mix Flexibility (PF): This reflects the variety of product types the system can produce without incurring additional costs or delays. It's an indicator of the system's ability to adapt to changes in production demands.
- Routing Flexibility (RF): This measures the average number of operations per machining center. It indicates the system's ability to switch production between different products and processes efficiently.

Model Selection:

- CCR Input Oriented Model
- CCR Output Oriented Model
- BCC Input Oriented Model
- BCC Output Oriented Model

The BCC Input-oriented model was chosen due to the company's current emphasis on cost reduction and resource conservation. An input-oriented approach focuses on minimizing input usage while still achieving the desired level of outputs. This aligns with our goal to identify areas where manufacturing cells can reduce resource consumption without compromising their productivity, which is critical under the company's strategy to enhance efficiency in a tightened economic landscape.

3 Recommendation



	Efficiency Analysis			
DA	BCC-Input	BCC-Output	CCR-Input	CCR-Output
A	1	1	1	1
B	1	1	1	1
C	0.770325736	1.228154054	0.673050398	1.485772839
D	0.798937961	1.062514252	0.784925325	1.274006543
E	1	1	1	1
F	1	1	1	1
G	0.987680371	1.008710993	0.970291208	1.030618429
H	0.704384468	1.265402084	0.663149693	1.507955158
I	1	1	1	1
J	1	1	1	1
K	1	1	1	1
L	0.93014893	1.051916231	0.924958253	1.081129875
M	1	1	1	1
N	0.674565025	1.180613457	0.639442357	1.56386262
O	1	1	1	1
P	0.891548504	1.04465011	0.851888126	1.173863057
Q	1	1	1	1
R	0.870810152	1.547945205	0.644474089	1.551652763

Figure 1: DEA Analysis Result

Cells Recommended for Closure:

Cells C, D, H, L, M, N, and R are identified as the least efficient units, with scores ranging from 0.6746 to 0.8708. Given their lower efficiency ratings and the strategic necessity to streamline operations, these cells are recommended for closure. Their resources could be better allocated to higher-performing cells to enhance overall division efficiency.

Efficiency Optimization for Remaining Cells:

Cells A, B, E, F, J, K, O, P, and Q all achieved perfect efficiency scores of 1. These cells should be maintained and possibly serve as benchmarks for best practices that can be adopted by other cells.

Cells G and P with scores just below 1 (0.9877 and 0.8915, respectively), are nearly optimal in their performance and should be reviewed to identify minor improvements that can elevate them to full efficiency.

4 Additional Analysis

1. *The company decides that no matter how difficult the economical times are for the company they will not fire any employees. How would you change your analysis to account for this factor?*

Answer: Under a no layoff policy, our analysis must shift from closing manufacturing cells to optimizing them. Efficiency improvements within the current cells would focus on reducing non-labor inputs and reallocating employees to maximize their skills and productivity within the efficient cells. We may need to consider:

- Cross-training employees to operate multiple types of machinery, which would enhance labor flexibility.
- Implementing job rotation programs to prevent skills from becoming obsolete and to stimulate employee engagement.
- Utilizing the employees from less efficient cells to support areas in efficient cells that may be understaffed or require additional skills.

The goal would be to preserve jobs while still achieving the necessary operational efficiency to remain competitive. This approach may alter the ranking of cells based on their ability to absorb additional labor without reducing efficiency scores.

2. *When this same set of data is run as a CCR-I model, all of the cells that are efficient in the original analysis remain efficient, why is this so and what does it state about your boss's original belief about returns to scale?*

Answer: The efficiency results from the CCR-I model, which assumes constant returns to scale, have affirmed the efficiency status of all the manufacturing cells previously identified as efficient under the BCC-I model. This alignment suggests that, within the observed data range, the efficient cells are likely operating at scales where increasing or decreasing inputs would lead to proportional changes in outputs, characteristic of constant returns to scale.

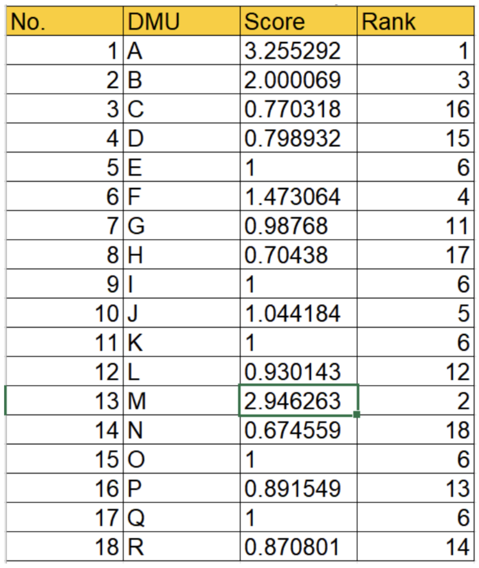
This observation has implications regarding the boss's original belief about returns to scale within the company. It suggests that for the efficient cells, the operations might actually be exhibiting constant returns to scale rather than variable. This could mean that these cells are already operating at an optimal scale, where their size and output levels are well-aligned. The efficient cells are utilizing their resources in such a way that any change in scale would still maintain their efficiency – a property not assumed in the variable returns to scale model.

Thus, the data may indicate a need to reevaluate the assumption of variable returns to scale for these particular cells, or at least to acknowledge that their current operation is consistent with constant returns to scale. This information could be pivotal for strategic planning, especially if the company considers expanding or contracting its operations.

3. *Although all of the efficient cells will have efficiency scores of 1, is there any particular cell that you would highly caution Harry from closing? Why?*

Answer: The use of the super-efficiency DEA model is instrumental in our context because it provides a means to DMUs beyond the standard efficiency frontier. By allowing efficiency scores greater than 1, the super-efficiency model identifies not just whether a unit is performing without waste, but also the extent to which it outperforms other efficient units. This is particularly crucial for strategic decision-making in contexts where multiple units are efficient, but resources are limited and some efficient units may still need to be closed.

Upon reviewing the super-efficiency results, it is evident that Cell A, with the highest score of 3.255292, stands out as exceptionally robust. It far surpasses the benchmark efficiency score, indicating it could potentially handle input reductions while maintaining its current level of output. In addition to Cell A, Cells M and B also showcase high super-efficiency scores of 2.946263 and 2.000069 respectively, indicating they too are significantly more efficient compared to their peers.



No.	DMU	Score	Rank
1	A	3.255292	1
2	B	2.000069	3
3	C	0.770318	16
4	D	0.798932	15
5	E	1	6
6	F	1.473064	4
7	G	0.98768	11
8	H	0.70438	17
9	I	1	6
10	J	1.044184	5
11	K	1	6
12	L	0.930143	12
13	M	2.946263	2
14	N	0.674559	18
15	O	1	6
16	P	0.891549	13
17	Q	1	6
18	R	0.870801	14

Figure 2: Super-Efficiency DEA Model Result

According to the super-efficiency scores, it is recommended that careful consideration be given before making any decision to close Cell A, which shows extraordinary efficiency. Similarly, although not quite as high as Cell A, Cells M and B should also be regarded as critical assets within the manufacturing system due to their super-efficient operations. These cells have proven their capability to operate effectively, even under potential input constraints, and closing them might lead to a loss of valuable production capacity and operational resilience. Preserving Cells A, M, and B could be essential to maintaining a strong and efficient production landscape, especially in an environment where efficiency and productivity are of utmost importance.

5 Appendix

Python Implementation for DEA model

```python
import gurobipy
import pandas as pd
class DEA(object):
    def __init__(self, DMUs_Name, X, Y):
        self.m1, self.m1_name, self.m2, self.m2_name= X.shape[1], X.columns.tolist(), Y.shape[1], Y.columns.tolist()
        self.DMUs, self.X, self.Y = gurobipy.multidict({DMU: [X.loc[DMU].tolist(), Y.loc[DMU].tolist()] for DMU in DMUs_Name})
        print(f'Running DEA Model')

    def __CCR_Input(self):
        for k in self.DMUs:
            MODEL = gurobipy.Model()
            # addVar() creates a single variable, addVars() creates multiple variables
            theta, lambdas = MODEL.addVar(), MODEL.addVars(self.DMUs)
            MODEL.update()
            MODEL.setObjective(theta, sense=gurobipy.GRB.MINIMIZE)
            MODEL.addConstrs(
                gurobipy.quicksum(lambdas[i] * self.X[i][j] for i in self.DMUs) <= theta * self.X[k][j] for j in range(self.m1))
            MODEL.addConstrs(
                gurobipy.quicksum(lambdas[i] * self.Y[i][j] for i in self.DMUs ) >= self.Y[k][j] for j in range(self.m2))
            for i in self.DMUs:
                lambdas[i].setAttr(gurobipy.GRB.Attr.LB, 0.0)
            MODEL.setParam('OutputFlag', 0)
            MODEL.optimize() # Execute the linear programming model
            # Record the objective value as the result for CCR efficiency
            self.Result.at[k, ('Efficiency Analysis', 'CCR-Input')] = MODEL.objVal
        return self.Result

    def __CCR_Output(self):
        for k in self.DMUs:
            MODEL = gurobipy.Model()
            # addVar() creates a single variable, addVars() creates multiple variables
            # In the output-oriented model, phi represents the efficiency score
            phi, lambdas = MODEL.addVar(lb=1.0, vtype=gurobipy.GRB.CONTINUOUS), MODEL.addVars(self.DMUs)
            MODEL.update()  ## Update variable environment
            MODEL.setObjective(phi, sense=gurobipy.GRB.MAXIMIZE)
            MODEL.addConstrs(
                gurobipy.quicksum(lambdas[i] * self.X[i][j] for i in self.DMUs) <= self.X[k][j] for j in range(self.m1))
            MODEL.addConstrs(
                gurobipy.quicksum(lambdas[i] * self.Y[i][j] for i in self.DMUs) >= phi * self.Y[k][j] for j in range(self.m2))
            for i in self.DMUs:
                lambdas[i].setAttr(gurobipy.GRB.Attr.LB, 0.0)
            MODEL.setParam('OutputFlag', 0)
            MODEL.optimize()
            self.Result.at[k, ('Efficiency Analysis', 'CCR-Output')] = MODEL.objVal if MODEL.status == gurobipy.GRB.Status.OPTIMAL else 'N/A'
        return self.Result

    def __BCC_Input(self):
        for k in self.DMUs:
            MODEL = gurobipy.Model()
            # addVar() creates a single variable, addVars() creates multiple variables
            theta, lambdas = MODEL.addVar(), MODEL.addVars(self.DMUs)
            MODEL.update()  ## Update variable environment
            MODEL.setObjective(theta, sense=gurobipy.GRB.MINIMIZE)
            # Create constraints
            MODEL.addConstrs(
                gurobipy.quicksum(lambdas[i] * self.X[i][j] for i in self.DMUs ) <= theta * self.X[k][j] for j in range(self.m1))
            MODEL.addConstrs(
                gurobipy.quicksum(lambdas[i] * self.Y[i][j] for i in self.DMUs ) >= self.Y[k][j] for j in range(self.m2))
            for i in self.DMUs:
                lambdas[i].setAttr(gurobipy.GRB.Attr.LB, 0.0)
            MODEL.addConstr(gurobipy.quicksum(lambdas[i] for i in self.DMUs) == 1, name='convexity_constraint')
            MODEL.setParam('OutputFlag', 0) # Turn off solving log; Hide the computation process, only show the final result.
            MODEL.optimize() # Execute the linear programming model
            self.Result.at[k, ('Efficiency Analysis', 'BCC-Input')] = MODEL.objVal if MODEL.status == gurobipy.GRB.Status.OPTIMAL else 'N/A'
        return self.Result

    def __BCC_Output(self):
        for k in self.DMUs:
            MODEL = gurobipy.Model()
            phi, lambdas = MODEL.addVar(vtype=gurobipy.GRB.CONTINUOUS, name='phi'), MODEL.addVars(self.DMUs)
            MODEL.update()
            MODEL.setObjective(phi, sense=gurobipy.GRB.MAXIMIZE)
            # Constraints for the BCC output-oriented model
            MODEL.addConstrs(
                (gurobipy.quicksum(lambdas[i] * self.X[i][j] for i in self.DMUs) <= self.X[k][j] for j in range(self.m1)),
                name='input_constraints')
            MODEL.addConstrs(
                (gurobipy.quicksum(lambdas[i] * self.Y[i][j] for i in self.DMUs) >= phi * self.Y[k][j] for j in range(self.m2)),
                name='output_constraints')
            # Convexity constraint: sum of lambdas should be 1
            MODEL.addConstr(gurobipy.quicksum(lambdas[i] for i in self.DMUs) == 1, name='convexity_constraint')
            MODEL.setParam('OutputFlag', 0)
            MODEL.optimize()
            self.Result.at[k, ('Efficiency Analysis', 'BCC-Output')] = MODEL.objVal if MODEL.status == gurobipy.GRB.Status.OPTIMAL else 'N/A'
        return self.Result

    def dea(self):
        # Define column groups for the DataFrame
        columns_Page = ['Efficiency Analysis'] * 4
        columns_Group = ['BCC-Input', 'BCC-Output', 'CCR-Input','CCR-Output']
        # Initialize the DataFrame to store results
        self.Result = pd.DataFrame(index=self.DMUs, columns=[columns_Page, columns_Group])
        self.__CCR_Input()
        self.__CCR_Output()
        self.__BCC_Input()
        self.__BCC_Output()
        return self.Result

    def analysis(self, file_name=None):
        Result = self.dea()
        file_name = 'DEA_Analysis_Results.xlsx'
        Result.to_excel(file_name, 'DEA_Analysis_Results')
```

Generating Reports

```
    if __name__ == '__main__':
    data = r"DEA_Bossaball_DataInputSheet.xlsx"  #Read Data
    x = pd.read_excel(data, sheet_name = 0, usecols = [1,2])  # Input Col
    y = pd.read_excel(data, sheet_name = 0, usecols = [3,4])  # Output Col
    state = pd.read_excel(data, sheet_name = 0, usecols = [0])
    dea = DEA(DMUs_Name= range(0,12), X=x, Y=y) # 12 teams in the original NBL dataset
    dea.analysis()
```