University of Waterloo

Department of Management Science

MSCI 434: Supply Chain Management

Spring 2023

Team 27 Final Report

Thomas Enns	20823674
Zuhayr Shaikh	20844653

Introduction

In this project, we will consider a model for optimal product mix in Material Resource Planning developed by Harish C. Bahl, Sharam Taj, and Wayne Corcoran in a paper titled "A linear-programming model formulation for optimal product-mix decisions in material requirements-planning environments". The paper addresses material-requirements-planning (MRP) environments, focusing on optimal product-mix decisions. By integrating linear programming (LP) techniques into the MRP system, the paper aims to enhance decision-making by maximizing organizational profitability. The objective is to determine the optimal production quantities for components, subassemblies, and final products, while considering various constraints such as production capacities, overtime utilization, and marketing requirements.

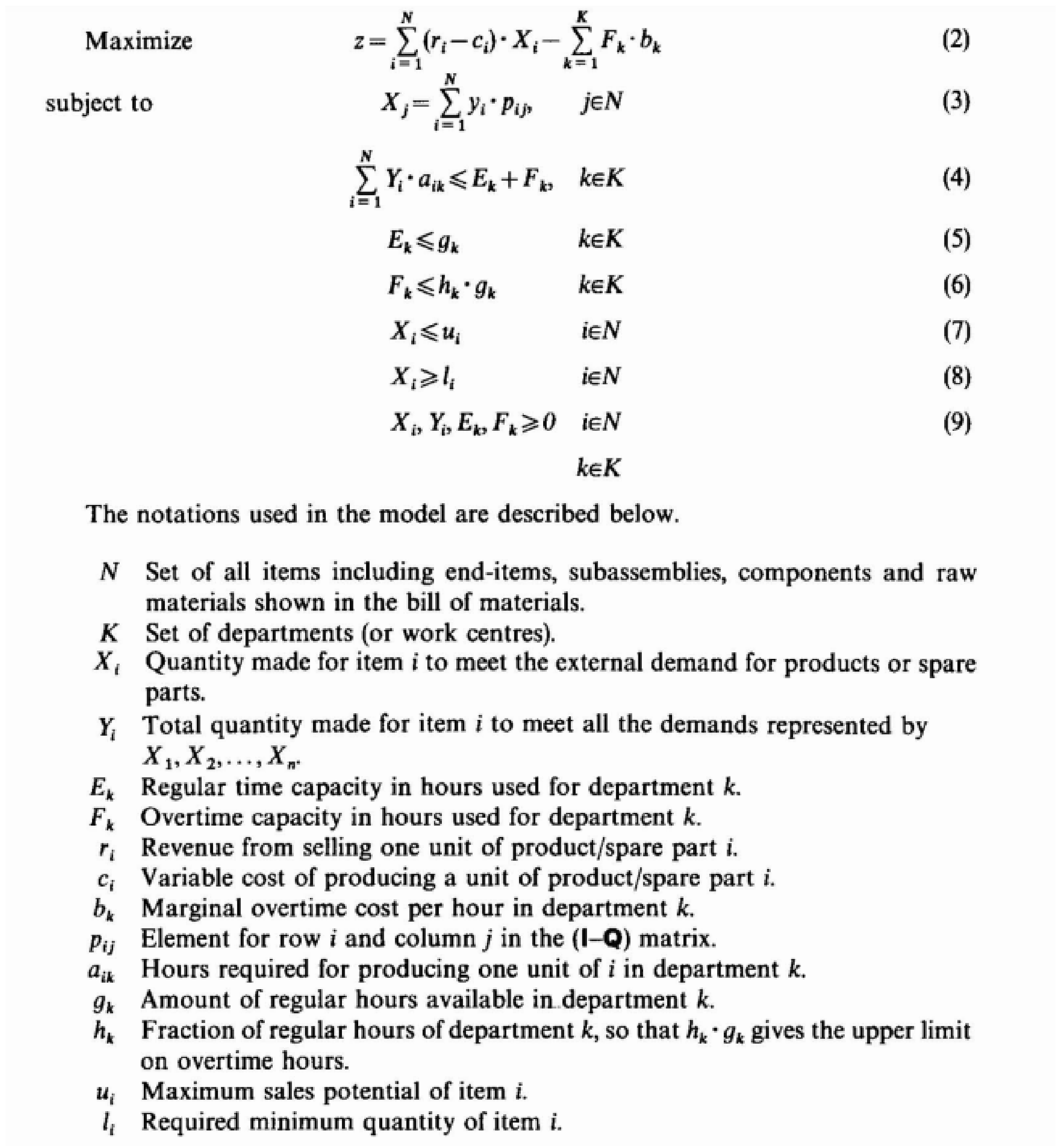
Optimal product mix decisions are crucial in MRP environments, ensuring efficient resource allocation, maximizing profits, and meeting customer demands. Bahl et al.'s model utilizes linear programming and BOMs to capture production complexities, enabling informed decisions for enhanced efficiency and increased profitability. Its computational efficiency and scalability make it a valuable tool for organizations seeking to optimize their product mix strategies, gaining a competitive edge in the market.

We first use Gurobi and Python to run their model, using the same fictional data as the researchers, to create a benchmark solution. We then run their model again, using an initial feasible solution, to compare solution times. Finally, we incorporate a penalty cost for failing to meet demand into the model, and apply it to a real company: Devinci Bicycles, based in

Chicoutimi QC, and three of their main products. We split their demand into periods to more accurately model the variable demand for this seasonal item, and solve for each period.

Model Description

The authors propose a mathematical model based on linear programming principles. The model incorporates the Bill of Materials (BOMs), which represents the product structure, and utilizes a quantity matrix to define the interrelationships between different items. By formulating the problem with linear programming, the authors effectively capture the complexities and constraints associated with production capacities, and overtime usage as seen in the model in Figure 1, which maximizes total profits while adhering to production capacity constraints:



Maximize $$z=\sum_{i=1}^{N}(r_i-c_i)\cdot X_i-\sum_{k=1}^{K}F_k\cdot b_k \qquad (2)$$

subject to $$X_j=\sum_{i=1}^{N}y_i\cdot p_{ij}, \quad j\in N \qquad (3)$$

$$\sum_{i=1}^{N}Y_i\cdot a_{ik}\leqslant E_k+F_k, \quad k\in K \qquad (4)$$

$$E_k\leqslant g_k \quad k\in K \qquad (5)$$

$$F_k\leqslant h_k\cdot g_k \quad k\in K \qquad (6)$$

$$X_i\leqslant u_i \quad i\in N \qquad (7)$$

$$X_i\geqslant l_i \quad i\in N \qquad (8)$$

$$X_i, Y_i, E_k, F_k\geqslant 0 \quad i\in N, \; k\in K \qquad (9)$$

The notations used in the model are described below.

N	Set of all items including end-items, subassemblies, components and raw materials shown in the bill of materials.
K	Set of departments (or work centres).
X_i	Quantity made for item i to meet the external demand for products or spare parts.
Y_i	Total quantity made for item i to meet all the demands represented by $X_1, X_2, \ldots, X_n$.
E_k	Regular time capacity in hours used for department k.
F_k	Overtime capacity in hours used for department k.
r_i	Revenue from selling one unit of product/spare part i.
c_i	Variable cost of producing a unit of product/spare part i.
b_k	Marginal overtime cost per hour in department k.
p_{ij}	Element for row i and column j in the $(\mathbf{I}-\mathbf{Q})$ matrix.
a_{ik}	Hours required for producing one unit of i in department k.
g_k	Amount of regular hours available in department k.
h_k	Fraction of regular hours of department k, so that $h_k\cdot g_k$ gives the upper limit on overtime hours.
u_i	Maximum sales potential of item i.
l_i	Required minimum quantity of item i.

Figure 1: Initial Model Formulation Bahl et Al. 1991)

The paper provides a hypothetical example to illustrate the application of the model and discusses the formulation and solution process using linear programming computer software. The example includes two products whose BOMs can be seen below in Figure 2:

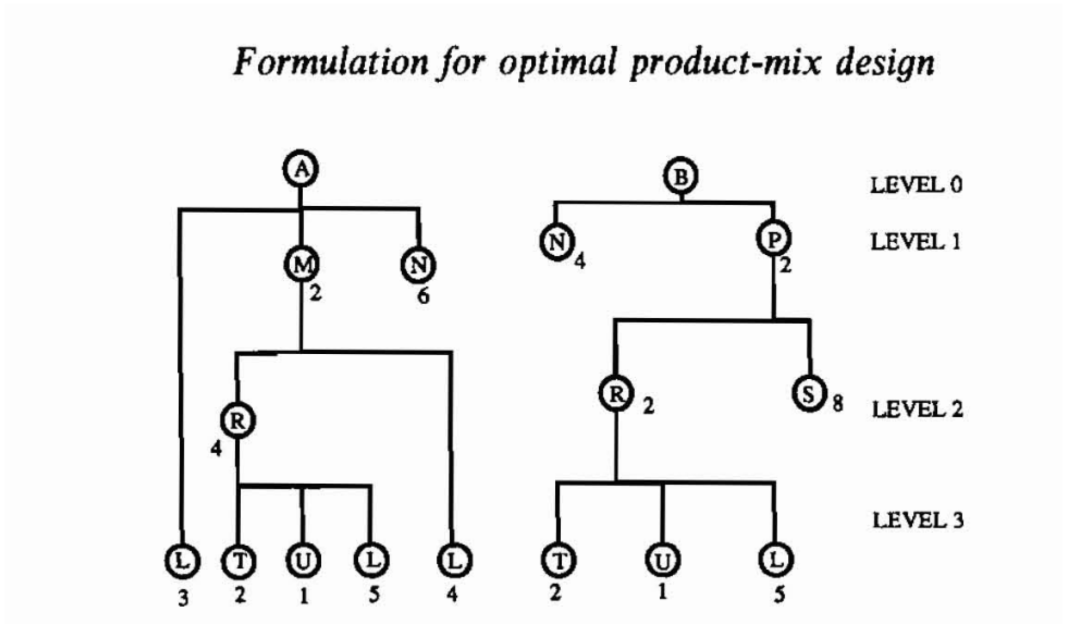
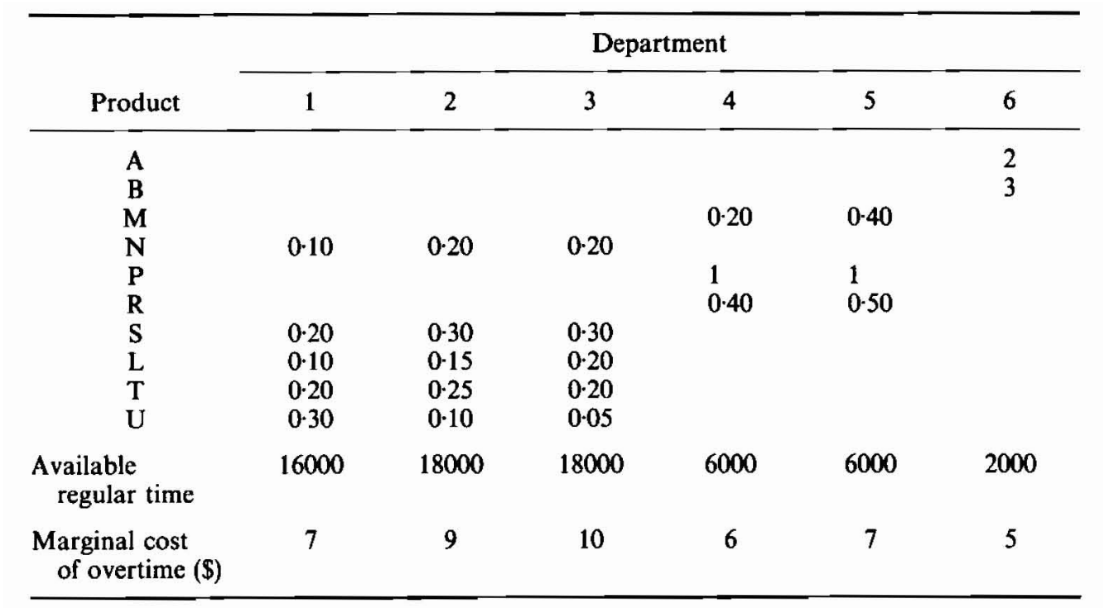


Figure 2: Bill of Materials for Fictional Bicycle Company (Bahl et Al. 1991)

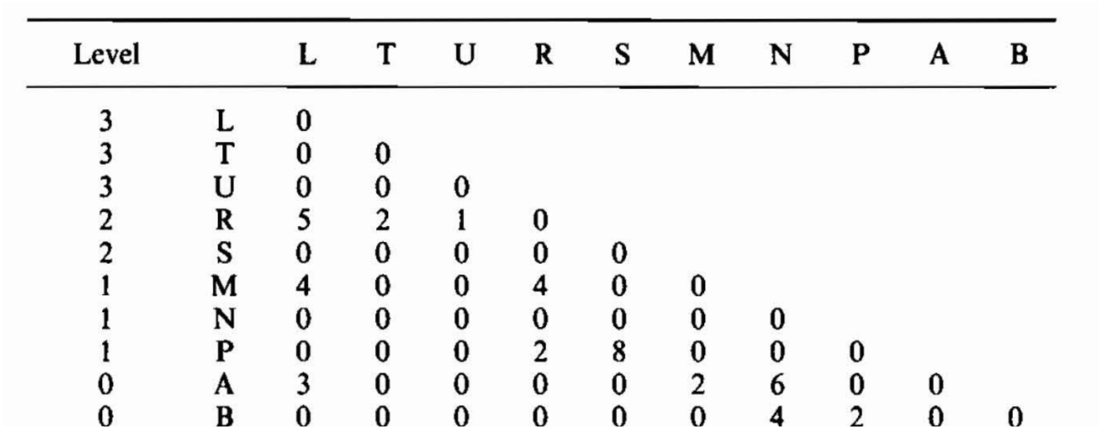
The model assumes that the "regular production capabilities are limited but can be increased in different departments by deploying overtime" (Bahl et Al. 1991). Each component in a BOM, requires a set amount of time from certain departments to create. Each department has a limited time budget and can deploy up to 25% more time in overtime which carries a marginal cost per unit-time. The time requirements can be seen below in Table 1.



Product	Department 1	2	3	4	5	6
A						2
B						3
M				0·20	0·40	
N	0·10	0·20	0·20			
P				1	1	
R				0·40	0·50	
S	0·20	0·30	0·30			
L	0·10	0·15	0·20			
T	0·20	0·25	0·20			
U	0·30	0·10	0·05			
Available regular time	16000	18000	18000	6000	6000	2000
Marginal cost of overtime ($)	7	9	10	6	7	5

Table 1: Time Allocated Per Department Per Product (Bahl et Al. 1991)

The authors introduce a concept known as the quantity matrix *Q*, based on the BOMs, where each component q_{ij} signifies the necessary amount of item *j* directly required to construct one unit of item *i*. They also apply a constraint where items are organized starting from the lowest level, proceeding to the subsequent higher levels. The structure of the quantity matrix related to the BOMs depicted in Figure 2 is exhibited in Table 2. The illustration makes it apparent that every row outlines the process of creating the item corresponding to that row. For instance, Table 2 indicates that the assembly of a single unit of *R* requires five units of *L*, two units of *T*, and one unit of *U*. Additionally, each column in the table demonstrates the specific location where the corresponding item is directly utilized in the assembly process.



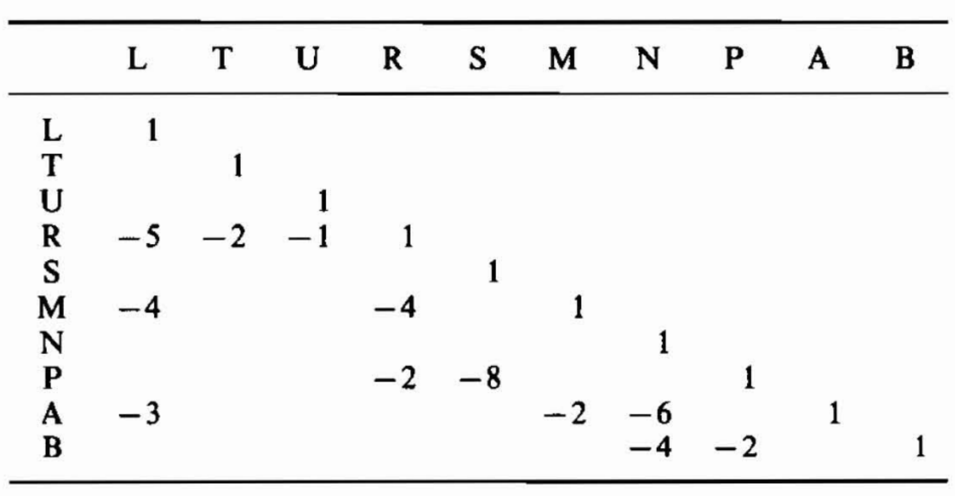
Level		L	T	U	R	S	M	N	P	A	B
3	L	0									
3	T	0	0								
3	U	0	0	0							
2	R	5	2	1	0						
2	S	0	0	0	0	0					
1	M	4	0	0	4	0	0				
1	N	0	0	0	0	0	0	0			
1	P	0	0	0	2	8	0	0	0		
0	A	3	0	0	0	0	2	6	0	0	
0	B	0	0	0	0	0	0	4	2	0	0

Table 2: Quantity Matrix, Q (Bahl et Al. 1991)

To denote the net availability of a subassembly for meeting outside demand after subtracting internal demand from total internal production of said assembly, the following equation is used to derive the constraint for each sub-assembly:

$$X = Y * (I - Q),$$

Where *I* represents the identity matrix. Figure 3 below depicts the (I-Q) matrix corresponding to the fictional example within the paper:



	L	T	U	R	S	M	N	P	A	B
L	1									
T		1								
U			1							
R	−5	−2	−1	1						
S					1					
M	−4			−4		1				
N							1			
P				−2	−8			1		
A	−3					−2	−6		1	
B							−4	−2		1

Figure 3: (I-Q) Matrix (Bahl et Al. 1991)

Model with Original Data

The code associated with this section can be found in Appendix A.

The model was first run with the 10 different components and 6 different departments described in the paper, using the associated data described previously. Please note that all the data except the sales price of items (revenue from sales) was available in the paper. Thus, for the baseline example, sales prices for each component were generated by multiplying the cost to manufacture these items by 1.5, assuming that their associated profit margin is 50% before sales costs. This first run yielded an objective function value of $225,021.67 and decision variable values similar to those in the paper. We attribute the differences in the optimal solution to our estimation of the sale price value of the items; we have no way of knowing the values originally used in the paper. This optimization ran in approximately 1 second, which is fast, but to be expected with such a small dataset.

We used the worst candidate solution from our initial optimization as an initial solution to speed up solving time. This predictably brought us back to the same optimal solution but in approximately 0 seconds of solving time. This was a marked improvement. Overall we can see that this model is quite computationally efficient, and although we cannot be sure as the price data was assumed, it is performing similarly to described in the literature.

Model and Data Updates

The code associated with this section can be found in Appendices B and C.

To incorporate a penalty cost for failing to meet demand, we declare a new parameter in addition to those described in the modeling section. This **new parameter $\boldsymbol{v_i}$** represents the cost of lost goodwill per unit of demand not met for a product. We then modify the objective function to account for the cost of lost goodwill when demand is not met. The modified objective function can be seen below:

$$\text{Maximize} \qquad Z = \sum_{i \in N} [X_i(r_i - c_i) - v_i(u_i - X_i)] - \sum_{k \in K} F_k b_k$$

We now attempt to apply this model to approximate decisions at the Devinci Bicycle Company in Chicoutimi Quebec. Although they manufacture several types of bicycles including e-bikes, the three products under consideration are:

- City - Cartier Urban (1000 CAD) - Product A
- Kids - EWOC Mountain (700 CAD) - Product B
- Mountain - Marshall Trail (3400 CAD) - Product C

We are assuming that all the components and sub-assemblies are also sold as spare parts. The (simplified) BOMs in Figure 4 are based on the Devinci product images (Devinci Bikes, n.d.) as well as bike assembly practices (Lu & Trappey, 2009).

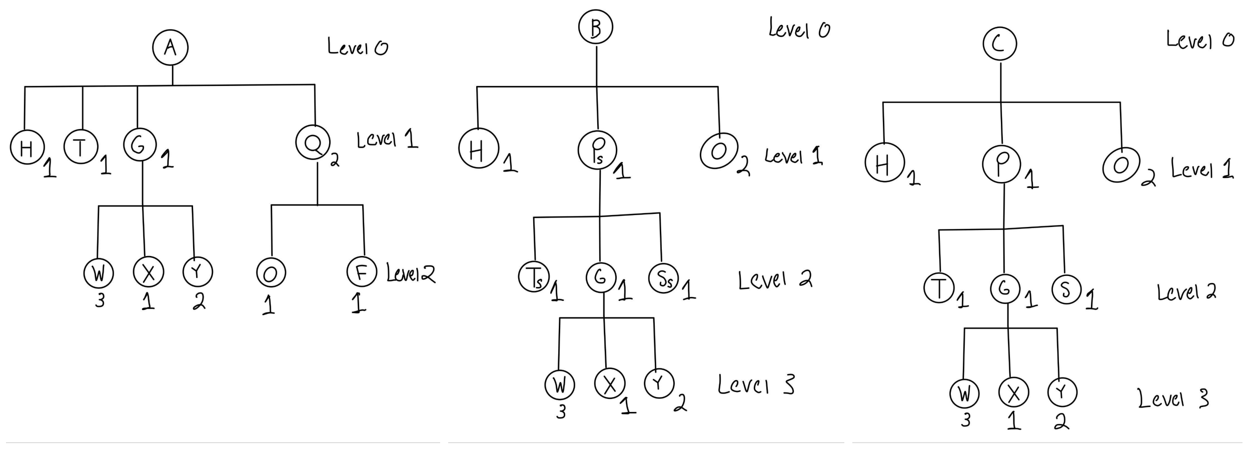


Figure 4: Devinci - BOMs

All the letters denoting products, parts, or sub-assemblies are defined below in Table 3. In lieu of real world sales prices and production cost data for the numerous components, prices are determined by our reasonable assumptions based on the available prices of the items in question.

We employed Python programming to estimate the seasonal demand for bicycles and spare parts, using a cosine function to capture the cyclical patterns of consumer demand throughout the year. The function's peak and trough values correspond to the highest (summer) and lowest (winter) demands, respectively. From these calculated monthly fluctuations, we generated summative values for both halves of the year, offering a compact yet effective model for operational planning and inventory management (refer to Appendix C).

Letter	**Definition**	**Demand**		**Sale Price ($)**	**Production Cost ($)**
		Jan. - Jun.	**Jun. - Dec.**		
A	Urban Bike	4796	5998	1000	800
B	Kids Bike	1996	2498	700	500
C	Mountain Bike	3196	3998	3400	2800
H	Handlebars	217	257	200	50
T	Basic Frame	477	537	700	400
T_s	Kids Basic Frame	58	86	400	200
G	Transmission	16	20	300	100
Q	Wheels with Fenders	0	0	0	80
P	Mountain Frame	436	518	2000	1600
P_s	Kids Mountain Frame	97	137	400	200
S	Suspension	357	417	75	40

S_s	Kids Suspension	25	41	50	25
O	Wheel	636	738	100	50
F	Fender	436	518	20	5
W	Gear	157	197	25	15
X	Chain	138	156	75	20
Y	Pedal	556	758	10	5

Table 3: Devinci - Components: Pricing, Cost, & Demand

Since we have introduced a cost of lost goodwill for failing to meet demand for a product, we must also estimate this cost for each final product and component. Failing to meet the demand for a spare part might not necessarily mean losing a customer, as they may have no other suppliers and will simply have to wait. Failing to meet demand for a bicycle might mean, however, is assumed to be more likely to increase the risk that demand goes elsewhere. Thus, we have set the penalty for failing to meet demand for a spare part at 5% of the sale price and the penalty for final products at 10% of the sale price.

As per the BOMs, the (I - Q) matrix for components was calculated and is displayed below in Table 4:

	X	**W**	**Y**	**T**	**G**	**S**	**Ts**	**Ss**	**O**	**F**	**P**	**H**	**Ps**	**Q**	**A**	**B**	**C**
X	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
G	**-1**	**-3**	**-2**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
Ts	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0

Ss	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
P	0	0	0	**-1**	**-1**	**-1**	0	0	0	0	**1**	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
Ps	0	0	0	0	**-1**	0	**-1**	**-1**	0	0	0	0	**1**	0	0	0	0
Q	0	0	0	0	0	0	0	0	**-1**	**-1**	0	0	0	**1**	0	0	0
A	0	0	0	**-1**	**-1**	0	0	0	0	0	0	**-1**	0	**-2**	**1**	0	0
B	0	0	0	0	0	0	0	0	**-2**	0	0	**-1**	**-1**	0	0	**1**	0
C	0	0	0	0	0	0	0	0	**-2**	0	**-1**	**-1**	0	0	0	0	**1**

Table 4: Devinci - (I - Q) Matrix

In our effort to understand Devinci's manufacturing process, we make an educated supposition about their internal operations. We divided the company into six integral departments: Material Handling & Storage (Department 1), Assembly - Other (Department 2), Assembly - Welding (Department 3), Quality Assurance (Department 4), Painting & Finishing (Department 5), and Packaging (Department 6).

Each of these departments plays a pivotal role in the overall production process. Material Handling & Storage focuses on the efficient movement and storage of materials, whereas the Assembly departments, whether Other or Welding, are tasked with the careful assembly of the various components that make up Devinci's products. Quality Assurance ensures that the assembled products meet the company's high-quality standards. The products are then moved to the Painting & Finishing department, where they are aesthetically enhanced and prepared for the final step, Packaging, where they are made ready for delivery or sale.

We further approximated the man-hours required for each product assembly based on an analysis of the Devinci sub-components. This was done using the product images available on their website, allowing us a closer look at the complexity and construction of each product. The estimations were grounded on a reasonable assumption about the time it would potentially take an experienced worker to assemble each product efficiently and accurately. However, it is important to note that these are estimated values and actual assembly times may vary depending on numerous factors like worker skill, equipment efficiency, and operational conditions.

Product	1 (Material Handling & Storage)	2 (Assembly - Other)	3 (Assembly - Welding)	4 (Quality Assurance)	5 (Painting & Finishing)	6 (Packaging)
X	0.05	0.2	0	0.05	0	0
W	0.05	0.3	0	0.1	0	0
Y	0.05	0.05	0	0.02	0	0
T	0	0.3	0.4	0.8	2	0
G	0	0.6	0	0.7	0.2	0
S	0	0.2	0	0.7	1	0
Ts	0	0.2	0.3	0.7	1.5	0
Ss	0	0.15	0	0.65	1	0
O	0	0.15	1	0.8	0.3	0
F	0	0.1	0	0.1	0.1	0
P	0	1	0.5	1	2	0
H	0	0.2	0.5	0.5	0	0
Ps	0	1	0.5	0.8	1.7	0
Q	0	0.4	1	1	0.5	0
A	0.5	1	1	1	1	0.3
B	0.5	1	1	1	1	0.3
C	0.3	1	0.8	1	1	0.2
Available Regular Time	4000	18000	17000	19000	19000	2000
Marginal Cost of Overtime ($)	10	20	30	15	15	10

Table 5: Devinci - Time Allocated Per Department Per Product

Based on the estimated values for the Devinci Bikes production and demand, we ran the optimization model with the new objective function. Since we needed to calculate the optimal product mix for both periods under consideration (first and second half of the year), we ran the model twice, once for the first period and once for the second. Code and results of both runs (including decision variable values) can be found in Appendix B. Based on these parameters, Devinci can make $1,366,001.13 of profit in the first half of the year and $1,393,835.29 in the second half of the year should they select the optimal product mixes suggested by our solution.

It is clear that these results are based on several assumptions and that it would be far more applicable to do this with internal company data. Further steps could be to more accurately apply this modeling while having access to the internal demand forecasting and cost figures of an organization. This would facilitate the validation of the results as they would be contextualized by real revenue, cost, and profit figures. Another possible added complexity would be to use more complex BOMs that actually encompass every aspect of a product. These would likely greatly increase computing time but would make the results much more applicable.

Conclusion

We have summarized, analyzed, validated, and attempted to apply the research and modeling of Harish C. Bahl, Sharam Taj, and Wayne Corcoran for optimal product mix decisions in MRP environments. Their model is computationally efficient and the example they provided can indeed be solved, although they failed to provide pricing data, so it had to be inferred. Starting with an initial feasible solution (such as current product mix strategy) does have a positive impact on computation, although it is not necessary. Incorporating a parameter for the cost of lost-goodwill for unmet demand and applying the modified model to a more complex example demonstrated the versatility of the researcher’s modeling. We applied the model for a larger product set and two different instances of estimated variable demand (refer to methodology within Appendix C) corresponding to different seasonal conditions for bicycle sales. While this model is seemingly quite an efficient tool to aid in MRP product mix decisions, it would be valuable to validate it with actual figures rather than estimations and observe how it performs.

References

Devinci Bikes. (n.d.). *Devinci bikes - proudly making bikes and e-bikes in Canada*. Devinci. https://www.devinci.com/en/

HARISH C. BAHL, SHARAM TAJ & WAYNE CORCORAN (1991) A linear programming model formulation for optimal product-mix decisions in material-requirements planning environments, THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH, 29:5, 1025-1034, https://doi.org/10.1080/00207549108930117

Lu, Tung-Hung & Trappey, Amy. (2009). Development of a Web-Based Mass Customization Platform for Bicycle Customization Services. 10.1007/978-1-84882-762-2_80.

Ralph Buehler & John Pucher (2021) COVID-19 Impacts on Cycling, 2019–2020, Transport Reviews, 41:4, 393-400, DOI: 10.1080/01441647.2021.1914900

Appendix A - Code & Output for Paper Data & Model

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd

!pip install gurobipy>=9.5.1
import gurobipy as gp
from gurobipy import GRB as GRB
import numpy as np
```

In [ ]:
```
# Create environment with WLS license
e = gp.Env(empty=True)
e.setParam('WLSACCESSID', '7e2d40a7-904b-4d00-b37c-6993c3716fb6')
e.setParam('WLSSECRET', '731bbd0f-37ee-4c88-9d28-c4f67b9c7952')
e.setParam('LICENSEID', 2396892)
e.start()

# Create the model within the Gurobi environment
model = gp.Model(env=e)
```

```
Set parameter WLSAccessID
Set parameter WLSSecret
Set parameter LicenseID to value 2396892
Academic license - for non-commercial use only - registered to tenns@uwaterloo.ca
```

Basic Model from Paper

In [ ]:
```
#Define sets & Params
N = ['L', 'T', 'U', 'R','S','M','N','P','A','B']
K = [1,2,3,4,5,6]
#Prices (r) are estimated based on costs as this data was not not available in the paper
r = {'L':5*1.5, 'T':4*1.5, 'U':3*1.5, 'R':40*1.5,'S':6*1.5,'M':200*1.5,'N':10*1.5,'P':100*1.5,
c = {'L':5, 'T':4, 'U':3, 'R':40,'S':6,'M':200,'N':10,'P':100,'A':400,'B':200}
b = {1:7,2:9,3:10,4:6,5:7,6:5}
p = {
    'L':{'L':1, 'T':0, 'U':0, 'R':0,'S':0,'M':0,'N':0,'P':0,'A':0,'B':0},
    'T':{'L':0, 'T':1, 'U':0, 'R':0,'S':0,'M':0,'N':0,'P':0,'A':0,'B':0},
    'U':{'L':0, 'T':0, 'U':1, 'R':0,'S':0,'M':0,'N':0,'P':0,'A':0,'B':0},
    'R':{'L':-5, 'T':-2, 'U':-1, 'R':1,'S':0,'M':0,'N':0,'P':0,'A':0,'B':0},
    'S':{'L':0, 'T':0, 'U':0, 'R':0,'S':1,'M':0,'N':0,'P':0,'A':0,'B':0},
    'M':{'L':-4, 'T':0, 'U':0, 'R':-4,'S':0,'M':1,'N':0,'P':0,'A':0,'B':0},
    'N':{'L':0, 'T':0, 'U':0, 'R':0,'S':0,'M':0,'N':1,'P':0,'A':0,'B':0},
    'P':{'L':0, 'T':0, 'U':0, 'R':-2,'S':-8,'M':0,'N':0,'P':1,'A':0,'B':0},
    'A':{'L':-3, 'T':0, 'U':0, 'R':0,'S':0,'M':-2,'N':-6,'P':0,'A':1,'B':0},
    'B':{'L':0, 'T':0, 'U':0, 'R':0,'S':0,'M':0,'N':-4,'P':-2,'A':0,'B':1},
}
a = {
    'L':{1:0.1,2:0.15,3:0.2,4:0,5:0,6:0},
    'T':{1:0.2,2:0.25,3:0.2,4:0,5:0,6:0},
    'U':{1:0.3,2:0.1,3:0.05,4:0,5:0,6:0},
    'R':{1:0,2:0,3:0,4:0.4,5:0.5,6:0},
    'S':{1:0.2,2:0.3,3:0.3,4:0,5:0,6:0},
    'M':{1:0,2:0,3:0,4:0.2,5:0.4,6:0},
    'N':{1:0.1,2:0.2,3:0.2,4:0,5:0,6:0},
    'P':{1:0,2:0,3:0,4:1,5:1,6:0},
    'A':{1:0,2:0,3:0,4:0,5:0,6:2},
    'B':{1:0,2:0,3:0,4:0,5:0,6:3},
}
g = {1:16000,2:18000,3:18000,4:6000,5:6000,6:2000}
h = {1:0.25,2:0.25,3:0.25,4:0.25,5:0.25,6:0.25}
u = {'L':4000, 'T':4000, 'U':4000, 'R':1000,'S':1000,'M':1000,'N':1000,'P':1000,'A':65535,'B':
l = {'L':100, 'T':100, 'U':100, 'R':100,'S':100,'M':100,'N':100,'P':100,'A':500,'B':500}
```

```
In [ ]: #Decision Variables
        X = model.addVars(N, vtype=GRB.INTEGER, name="X")
        Y = model.addVars(N, vtype=GRB.INTEGER, name="Y")
        E = model.addVars(K, vtype=GRB.CONTINUOUS, name="E")
        F = model.addVars(K, vtype=GRB.CONTINUOUS, name="F")

        #Constraints
        [model.addConstr(X[j] == sum(Y[i]*p[i][j] for i in N)) for j in N]
        [model.addConstr(sum(Y[i]*a[i][k] for i in N) <= E[k]+F[k]) for k in K]
        [model.addConstr(E[k] <= g[k]) for k in K]
        [model.addConstr(F[k] <= h[k]*g[k]) for k in K]
        [model.addConstr(X[i] <= u[i]) for i in N]
        [model.addConstr(X[i] >= l[i]) for i in N]
        [model.addConstr(X[i] >= 0) for i in N]
        [model.addConstr(Y[i] >= 0) for i in N]
        [model.addConstr(E[k] >= 0) for k in K]
        [model.addConstr(F[k] >= 0) for k in K]

        #Set Objective
        model.setObjective(sum((r[i]-c[i])*X[i] for i in N) - sum(F[k]*b[k] for k in K), sense=GRB.MAXI

        # Keep more than best solution
        model.setParam(GRB.Param.PoolSolutions, 2)

        model.optimize()
        model.printAttr('X')
```

```
Set parameter PoolSolutions to value 2
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to tenns@uwaterloo.ca
Optimize a model with 80 rows, 32 columns and 131 nonzeros
Model fingerprint: 0x1f8e20aa
Variable types: 12 continuous, 20 integer (0 binary)
Coefficient statistics:
  Matrix range     [5e-02, 8e+00]
  Objective range  [2e+00, 2e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+02, 7e+04]
Presolve removed 73 rows and 17 columns
Presolve time: 0.00s
Presolved: 7 rows, 15 columns, 40 nonzeros
Variable types: 0 continuous, 15 integer (0 binary)
Found heuristic solution: objective 186752.50000

Root relaxation: objective 2.250217e+05, 3 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 225021.667    0    2 186752.500 225021.667  20.5%     -    0s
H    0     0                    225017.35000 225021.667  0.00%     -    0s

Explored 1 nodes (3 simplex iterations) in 0.06 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 2: 225017 186752

Optimal solution found (tolerance 1.00e-04)
Best objective 2.250173500000e+05, best bound 2.250216666667e+05, gap 0.0019%

    Variable            X
-------------------------
        X[L]         4000
        X[T]          100
        X[U]         3997
        X[R]          101
        X[S]          100
        X[M]          629
        X[N]         1000
        X[P]          100
        X[A]          500
        X[B]          500
        Y[L]        56101
        Y[T]        17734
        Y[U]        12814
        Y[R]         8817
        Y[S]         8900
        Y[M]         1629
        Y[N]         6000
        Y[P]         1100
        Y[A]          500
        Y[B]          500
        E[1]        16000
        E[2]        18000
        E[3]        18000
        E[4]         6000
        E[5]         6000
        E[6]         2000
```

```
F[2]          0.05
F[3]        1277.7
F[5]         160.1
F[6]           500
```

Find the worst feasible solution generated, To use as a starting point for optimization.

```
In [ ]:
nSolutions = model.SolCount
print('Number of solutions: ', nSolutions)
worst_obj = float('inf')
worst_sol = -1
# Loop through each solution and find worst one
for i in range(nSolutions):
    model.setParam(GRB.Param.SolutionNumber, i)
    obj = model.PoolObjVal
    if obj < worst_obj:
        worst_obj = obj
        worst_sol = i
model.setParam(GRB.Param.SolutionNumber, worst_sol)

# Print the worst solution
X_val = model.getAttr('Xn', X)
Y_val = model.getAttr('Xn', Y)
E_val = model.getAttr('Xn', E)
F_val = model.getAttr('Xn', F)
print('Worst Solution', worst_sol)
print('Objective value', worst_obj)
print('X values', X_val)
print('Y values', Y_val)
print('E values', E_val)
print('F values', F_val)
```

```
Number of solutions:  2
Worst Solution 1
Objective value 186752.5
X values {'L': 101.0, 'T': 4000.0, 'U': 4000.0, 'R': 100.0, 'S': 1000.0, 'M': 100.0, 'N': 100
0.0, 'P': 100.0, 'A': 500.0, 'B': 500.0}
Y values {'L': 39501.0, 'T': 17400.0, 'U': 10700.0, 'R': 6700.0, 'S': 9800.0, 'M': 1100.0,
'N': 6000.0, 'P': 1100.0, 'A': 500.0, 'B': 500.0}
E values {1: 16000.0, 2: 18000.0, 3: 18000.0, 4: 6000.0, 5: 6000.0, 6: 2000.0}
F values {1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 500.0}
```

Run with worst solution as Initial Feasible Solution

```
In [10]:
# Set worst solution as starting point for next optimization
model.reset()
for i in N:
    X[i].start = X_val[i]
    Y[i].start = Y_val[i]
for k in K:
    E[k].start = E_val[k]
    F[k].start = F_val[k]

model.optimize()
model.printAttr('X')
```

```
Discarded solution information
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to tenns@uwaterloo.ca
Optimize a model with 80 rows, 32 columns and 131 nonzeros
Model fingerprint: 0xe4cf776a
Variable types: 12 continuous, 20 integer (0 binary)
Coefficient statistics:
  Matrix range     [5e-02, 8e+00]
  Objective range  [2e+00, 2e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+02, 7e+04]

Loaded user MIP start with objective 186752

Presolve removed 73 rows and 17 columns
Presolve time: 0.00s
Presolved: 7 rows, 15 columns, 40 nonzeros
Variable types: 0 continuous, 15 integer (0 binary)
Found heuristic solution: objective 186755.00000

Root relaxation: objective 2.250217e+05, 3 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 225021.667    0    2 186755.000 225021.667  20.5%     -    0s
H    0     0                    225017.35000 225021.667  0.00%     -    0s

Explored 1 nodes (3 simplex iterations) in 0.04 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 2: 225017 186755

Optimal solution found (tolerance 1.00e-04)
Best objective 2.250173500000e+05, best bound 2.250216666667e+05, gap 0.0019%

    Variable            X
-------------------------
        X[L]         4000
        X[T]          100
        X[U]         3997
        X[R]          101
        X[S]          100
        X[M]          629
        X[N]         1000
        X[P]          100
        X[A]          500
        X[B]          500
        Y[L]        56101
        Y[T]        17734
        Y[U]        12814
        Y[R]         8817
        Y[S]         8900
        Y[M]         1629
        Y[N]         6000
        Y[P]         1100
        Y[A]          500
        Y[B]          500
        E[1]        16000
        E[2]        18000
        E[3]        18000
```

```
E[4]         6000
E[5]         6000
E[6]         2000
F[2]         0.05
F[3]       1277.7
F[5]        160.1
F[6]          500
```

Appendix B - Code & Output for Expanded Model & DeVinci Case

```
In [ ]: import matplotlib.pyplot as plt
        import pandas as pd

        !pip install gurobipy>=9.5.1
        import gurobipy as gp
        from gurobipy import GRB as GRB
        import numpy as np
```

```
In [ ]: # Create environment with WLS license
        e = gp.Env(empty=True)
        e.setParam('WLSACCESSID', '7e2d40a7-904b-4d00-b37c-6993c3716fb6')
        e.setParam('WLSSECRET', '731bbd0f-37ee-4c88-9d28-c4f67b9c7952')
        e.setParam('LICENSEID', 2396892)
        e.start()

        # Create the model within the Gurobi environment
        model = gp.Model(env=e)
        model2 = gp.Model(env=e)
```

```
Set parameter WLSAccessID
Set parameter WLSSecret
Set parameter LicenseID to value 2396892
Academic license - for non-commercial use only - registered to tenns@uwaterloo.ca
```

Expanded Model with DeVinci Data

```
In [ ]: #Define sets & Params
        N = ['X', 'W', 'Y', 'T','G','S','T_s','S_s','O','F','P','H','P_s','Q','A','B','C']
        K = [1,2,3,4,5,6]
        r = {
            'A': 1000,
            'B': 700,
            'C': 3400,
            'H': 200,
            'T': 700,
            'T_s': 400,
            'G': 300,
            'Q': 0,
            'P': 2000,
            'P_s': 400,
            'S': 75,
            'S_s': 50,
            'O': 100,
            'F': 20,
            'W': 25,
            'X': 75,
            'Y': 10
        }
        v = {
            'A': 50,
            'B': 50,
            'C': 50,
            'H': 5,
            'T': 20,
            'T_s': 15,
            'G': 10,
            'Q': 0,
            'P': 30,
            'P_s': 10,
            'S': 3,
            'S_s': 2,
            'O': 5,
            'F': 0,
    'W': 1,
    'X': 3,
    'Y': 0
}
c = {
    'A': 800,
    'B': 500,
    'C': 2800,
    'H': 50,
    'T': 400,
    'T_s': 200,
    'G': 100,
    'Q': 80,
    'P': 1600,
    'P_s': 200,
    'S': 40,
    'S_s': 25,
    'O': 50,
    'F': 5,
    'W': 15,
    'X': 20,
    'Y': 5
}
b = {1:10,2:20,3:30,4:15,5:15,6:10}
p = {
    'X':  {'X':1, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0, 'H
    'W':  {'X':0, 'W':1, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0, 'H
    'Y':  {'X':0, 'W':0, 'Y':1, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0, 'H
    'T':  {'X':0, 'W':0, 'Y':0, 'T':1, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0, 'H
    'G':  {'X':-1,'W':-3,'Y':-2,'T':0, 'G':1, 'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0, 'H
    'S':  {'X':0, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':1, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0, 'H
    'T_s':{'X':0, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':1,'S_s':0, 'O':0, 'F':0, 'P':0, 'H'
    'S_s':{'X':0, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':1,'O':0, 'F':0, 'P':0, 'H'
    'O':  {'X':0, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':1, 'F':0, 'P':0, 'H
    'F':  {'X':0, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':1, 'P':0, 'H
    'P':  {'X':0, 'W':0, 'Y':0, 'T':-1,'G':-1,'S':-1,'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':1, 'H
    'H':  {'X':0, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0, 'H
    'P_s':{'X':0, 'W':0, 'Y':0, 'T':0, 'G':-1,'S':0, 'T_s':-1,'S_s':-1,'O':0, 'F':0, 'P':0, 'H
    'Q':  {'X':0, 'W':0, 'Y':0, 'T':0, 'G':0, 'S':0, 'T_s':0, 'S_s':0, 'O':-1,'F':-1,'P':0, 'H
    'A':  {'X':0, 'W':0, 'Y':0, 'T':-1,'G':-1,'S':0, 'T_s':0, 'S_s':0, 'O':0, 'F':0, 'P':0,'H'
    'B':  {'X':0, 'W':0, 'Y':0, 'T':0,'G':0,'S':0, 'T_s':0, 'S_s':0, 'O':-2, 'F':0, 'P':0,'H':-
    'C':  {'X':0, 'W':0, 'Y':0, 'T':0,'G':0,'S':0,'T_s':0,'S_s':0,'O':-2,'F':0,'P':-1,'H':-1,'
}

a = {
    'X': {1: 0.05, 2: 0.2, 3: 0, 4: 0.05, 5: 0, 6: 0},
    'W': {1: 0.05, 2: 0.3, 3: 0, 4: 0.1, 5: 0, 6: 0},
    'Y': {1: 0.05, 2: 0.05, 3: 0, 4: 0.02, 5: 0, 6: 0},
    'T': {1: 0, 2: 0.3, 3: 0.4, 4: 0.8, 5: 2, 6: 0},
    'G': {1: 0, 2: 0.6, 3: 0, 4: 0.7, 5: 0.2, 6: 0},
    'S': {1: 0, 2: 0.2, 3: 0, 4: 0.7, 5: 1, 6: 0},
    'T_s': {1: 0, 2: 0.2, 3: 0.3, 4: 0.7, 5: 1.5, 6: 0},
    'S_s': {1: 0, 2: 0.15, 3: 0, 4: 0.65, 5: 1, 6: 0},
    'O': {1: 0, 2: 0.15, 3: 1, 4: 0.8, 5: 0.3, 6: 0},
    'F': {1: 0, 2: 0.1, 3: 0, 4: 0.1, 5: 0.1, 6: 0},
    'P': {1: 0, 2: 1, 3: 0.5, 4: 1, 5: 2, 6: 0},
    'H': {1: 0, 2: 0.2, 3: 0.5, 4: 0.5, 5: 0, 6: 0},
    'P_s': {1: 0, 2: 1, 3: 0.5, 4: 0.8, 5: 1.7, 6: 0},
    'Q': {1: 0, 2: 0.4, 3: 1, 4: 1, 5: 0.5, 6: 0},
    'A': {1: 0.5, 2: 1, 3: 1, 4: 1, 5: 1, 6: 0.3},
    'B': {1: 0.5, 2: 1, 3: 1, 4: 1, 5: 1, 6: 0.3},
    'C': {1: 0.3, 2: 1, 3: 0.8, 4: 1, 5: 1, 6: 0.2},
}

g = {1:4000,2:18000,3:17000,4:19000,5:19000,6:2000}
```

```
h = {1:0.25,2:0.25,3:0.25,4:0.25,5:0.25,6:0.25}
u = {
    'A': 4796,
    'B': 1996,
    'C': 3196,
    'H': 217,
    'T': 477,
    'T_s': 58,
    'G': 16,
    'Q': 0,
    'P': 436,
    'P_s': 97,
    'S': 357,
    'S_s': 25,
    'O': 636,
    'F': 436,
    'W': 157,
    'X': 138,
    'Y': 556
}
#For second period of year
# u = {
#     'A': 5998,
#     'B': 2498,
#     'C': 3998,
#     'H': 257,
#     'T': 537,
#     'T_s': 86,
#     'G': 20,
#     'Q': 0,
#     'P': 518,
#     'P_s': 137,
#     'S': 417,
#     'S_s': 41,
#     'O': 738,
#     'F': 518,
#     'W': 197,
#     'X': 156,
#     'Y': 758
# }
l = {
    'A': 1000,
    'B': 1000,
    'C': 1000,
    'H': 40,
    'T': 50,
    'T_s': 10,
    'G': 2,
    'Q': 0,
    'P': 50,
    'P_s': 87,
    'S': 70,
    'S_s': 5,
    'O': 120,
    'F': 60,
    'W': 20,
    'X': 20,
    'Y': 20
}
```

:

```
#Decision Variables
X = model.addVars(N, vtype=GRB.INTEGER, name="X")
Y = model.addVars(N, vtype=GRB.INTEGER, name="Y")
E = model.addVars(K, vtype=GRB.CONTINUOUS, name="E")
F = model.addVars(K, vtype=GRB.CONTINUOUS, name="F")
```

```
#Constraints
[model.addConstr(X[j] == sum(Y[i]*p[i][j] for i in N)) for j in N]
[model.addConstr(sum(Y[i]*a[i][k] for i in N) <= E[k]+F[k]) for k in K]
[model.addConstr(E[k] <= g[k]) for k in K]
[model.addConstr(F[k] <= h[k]*g[k]) for k in K]
[model.addConstr(X[i] <= u[i]) for i in N]
[model.addConstr(X[i] >= l[i]) for i in N]
[model.addConstr(X[i] >= 0) for i in N]
[model.addConstr(Y[i] >= 0) for i in N]
[model.addConstr(E[k] >= 0) for k in K]
[model.addConstr(F[k] >= 0) for k in K]

#Set Objective
model.setObjective(sum((r[i]-c[i])*X[i] for i in N) - sum(F[k]*b[k] for k in K), sense=GRB.MAX

model.optimize()

model.printAttr('X')
```

```
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to tenns@uwaterloo.ca
Optimize a model with 115 rows, 46 columns and 225 nonzeros
Model fingerprint: 0x82a2d657
Variable types: 12 continuous, 34 integer (0 binary)
Coefficient statistics:
  Matrix range     [2e-02, 3e+00]
  Objective range  [5e+00, 6e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 2e+04]
Presolve removed 104 rows and 19 columns
Presolve time: 0.00s
Presolved: 11 rows, 27 columns, 85 nonzeros
Variable types: 0 continuous, 27 integer (0 binary)
Found heuristic solution: objective 1043172.5500
Found heuristic solution: objective 1060487.0500

Root relaxation: objective 1.366001e+06, 12 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1366001.13    0    7 1060487.05 1366001.13  28.8%     -    0s
H    0     0                    1365960.4500 1366001.13  0.00%     -    0s

Explored 1 nodes (12 simplex iterations) in 0.03 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 3: 1.36596e+06 1.06049e+06 1.04317e+06

Optimal solution found (tolerance 1.00e-04)
Best objective 1.365960450000e+06, best bound 1.366001125561e+06, gap 0.0030%

    Variable            X
-------------------------
        X[X]          138
        X[W]          156
        X[Y]          556
        X[T]          477
        X[G]           16
        X[S]           70
      X[T_s]           58
      X[S_s]            5
        X[O]          120
        X[F]          430
        X[P]          436
        X[H]          217
      X[P_s]           87
        X[A]         1000
        X[B]         1000
        X[C]         1136
        Y[X]         3813
        Y[W]        11181
        Y[Y]         7906
        Y[T]         3049
        Y[G]         3675
        Y[S]         1642
      Y[T_s]         1145
      Y[S_s]         1092
        Y[O]         6392
        Y[F]         2430
```

```
       Y[P]         1572
       Y[H]         3353
     Y[P_s]         1087
       Y[Q]         2000
       Y[A]         1000
       Y[B]         1000
       Y[C]         1136
       E[1]         4000
       E[2]        18000
       E[3]        17000
       E[4]        19000
       E[5]        19000
       E[6]         2000
       F[4]      4749.97
       F[5]         3573
```

Second Half of the Year

In [ ]:
```python
# For second period of year
u = {
    'A': 5998,
    'B': 2498,
    'C': 3998,
    'H': 257,
    'T': 537,
    'T_s': 86,
    'G': 20,
    'Q': 0,
    'P': 518,
    'P_s': 137,
    'S': 417,
    'S_s': 41,
    'O': 738,
    'F': 518,
    'W': 197,
    'X': 156,
    'Y': 758
}

#Decision Variables
X = model2.addVars(N, vtype=GRB.INTEGER, name="X")
Y = model2.addVars(N, vtype=GRB.INTEGER, name="Y")
E = model2.addVars(K, vtype=GRB.CONTINUOUS, name="E")
F = model2.addVars(K, vtype=GRB.CONTINUOUS, name="F")

#Constraints
[model2.addConstr(X[j] == sum(Y[i]*p[i][j] for i in N)) for j in N]
[model2.addConstr(sum(Y[i]*a[i][k] for i in N) <= E[k]+F[k]) for k in K]
[model2.addConstr(E[k] <= g[k]) for k in K]
[model2.addConstr(F[k] <= h[k]*g[k]) for k in K]
[model2.addConstr(X[i] <= u[i]) for i in N]
[model2.addConstr(X[i] >= l[i]) for i in N]
[model2.addConstr(X[i] >= 0) for i in N]
[model2.addConstr(Y[i] >= 0) for i in N]
[model2.addConstr(E[k] >= 0) for k in K]
[model2.addConstr(F[k] >= 0) for k in K]

#Set Objective
model2.setObjective(sum((r[i]-c[i])*X[i] for i in N) - sum(F[k]*b[k] for k in K), sense=GRB.MA

model2.optimize()

model2.printAttr('X')
```

```
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to tenns@uwaterloo.ca
Optimize a model with 115 rows, 46 columns and 225 nonzeros
Model fingerprint: 0x6a5386cd
Variable types: 12 continuous, 34 integer (0 binary)
Coefficient statistics:
  Matrix range     [2e-02, 3e+00]
  Objective range  [5e+00, 6e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 2e+04]
Presolve removed 104 rows and 19 columns
Presolve time: 0.00s
Presolved: 11 rows, 27 columns, 85 nonzeros
Variable types: 0 continuous, 27 integer (0 binary)
Found heuristic solution: objective 1043172.5500
Found heuristic solution: objective 1055456.2000
Found heuristic solution: objective 1062159.4500

Root relaxation: objective 1.393835e+06, 14 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1393835.29    0    7 1062159.45 1393835.29  31.2%     -    0s
H    0     0                    1393666.2500 1393835.29  0.01%     -    0s
H    0     0                    1393711.5000 1393835.29  0.01%     -    0s

Explored 1 nodes (14 simplex iterations) in 0.18 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 5: 1.39371e+06 1.39367e+06 1.06216e+06 ... 1.04317e+06

Optimal solution found (tolerance 1.00e-04)
Best objective 1.393711500000e+06, best bound 1.393835291480e+06, gap 0.0089%

    Variable            X 
-------------------------
        X[X]          156 
        X[W]          196 
        X[Y]          758 
        X[T]          537 
        X[G]           20 
        X[S]           70 
      X[T_s]           86 
      X[S_s]            5 
        X[O]          121 
        X[F]          518 
        X[P]          518 
        X[H]          257 
      X[P_s]           88 
        X[A]         1000 
        X[B]         1000 
        X[C]         1075 
        Y[X]         3857 
        Y[W]        11299 
        Y[Y]         8160 
        Y[T]         3130 
        Y[G]         3701 
        Y[S]         1663 
      Y[T_s]         1174 
      Y[S_s]         1093 
  Y[O]         6271
  Y[F]         2518
  Y[P]         1593
  Y[H]         3332
Y[P_s]         1088
  Y[Q]         2000
  Y[A]         1000
  Y[B]         1000
  Y[C]         1075
  E[1]         4000
  E[2]        18000
  E[3]        17000
  E[4]        19000
  E[5]        19000
  E[6]         2000
  F[4]         4750
  F[5]       3760.9
```

Appendix C - Demand Seasonality Estimation

Our methodology to estimate seasonal demand for bicycles and spare parts adopts a blend of periodic function modeling and sum totals. The reason behind this approach is to embody the cyclical pattern of consumer demand throughout the year. We used the cosine function, a periodic function, to simulate these cycles of demand. The choice of the cosine function is due to its inherent periodicity, which naturally models the recurrent peaks and troughs in demand over the course of a year.In our approach, the maximum and minimum values of the cosine function represent the maximum (typically in summer) and minimum (usually in winter) demands. These were defined based on reasonable assumptions considering the seasonality in the usage of bicycles and spare parts.By applying the cosine function to each month, we obtained a fluctuating monthly demand pattern for each product. This gives us a dynamic demand model that effectively mirrors real-world demand trends.

Lastly, to assess overall seasonal demand, we summed the monthly demands for each product for the first half of the year (summer) and the second half (winter). This aggregation allowed us to understand the broad trends of demand over these periods, providing crucial insights for operational decisions such as production planning and inventory management.

```
import numpy as np

# Define the demand function
def demand(D_max, D_min, month):
    amplitude = (D_max - D_min) / 2
    vertical_shift = D_min + amplitude
    horizontal_shift = 6  # we set the max to June (month 6)
    # Convert month to the equivalent point in the cosine cycle (in
radians)
    radian_month = (month / 12) * (2 * np.pi)
    demand = vertical_shift + amplitude * np.cos(radian_month - 
horizontal_shift)
    return int(demand)  # Return the demand as an integer

# Define the product data
products = {
    "Urban Bike": (1200, 600),
    "Mountain Bike": (800, 400),
    "Kids Bike": (500, 250),
    "Handlebars": (50, 30),
    "Basic Frame": (100, 70),
    "Kids Basic Frame": (20, 5),
    "Transmission": (5, 2),
    "Wheels with Fenders": (0, 0),
    "Mountain Frame": (100, 60),
    "Kids Mountain Frame": (30, 10),
    "Suspension": (80, 50),
    "Kids Suspension": (10, 2),
    "Wheel": (140, 90),
    "Fender": (100, 60),
    "Gear": (40, 20),
    "Chain": (30, 20),
    "Pedal": (160, 60),
}

# Loop through each product and calculate the half-yearly demands
for product, (D_max, D_min) in products.items():
    demand_first_half = 0
    demand_second_half = 0
    for month in range(1, 13):
        monthly_demand = demand(D_max, D_min, month)
        if month <= 6:
            demand_first_half += monthly_demand
        else:
            demand_second_half += monthly_demand
    print(f"Demand for {product}:")
    print(f"Second half of the year: {demand_second_half}")
    print(f"First half of the year: {demand_first_half}")
```