OPERATIONS RESEARCH PROJECT REPORT

TOPIC: In our project we have discussed the industrial applications of operations research, specifically focusing on the multiperiod production planning.

TEAM:

1. BANDARI SHIKHARA
2. BHAVYA CHARITHA
3. SALONI KUMARI
4. ARCHIE AVIRATI
5. TRIPTI ROY

MULTIPERIOD PRODUCTION PLANNING:

A well-constructed production plan can help to boost revenue, profit and customer satisfaction, while a poorly designed plan can cause production problems and perhaps even sink the company. Specific benefits of production planning include:

- **Knowledge-** A production plan provides a framework for understanding the resources and production steps required to meet customer needs. It also helps companies understand the potential problems that may occur during production and how to mitigate them.
- **Efficiency-** Detailed production planning reduces bottlenecks and helps minimize costs. It also helps ensure the high quality of a product, and it keeps expenses on budget.
- **Customer satisfaction-** Production planning helps ensure that the company can make and deliver products to customers on time, leading to higher customer satisfaction and a greater likelihood of repeat business.

PROBLEM STATEMENT:

- The problem is to develop a mathematical optimization model for production planning and inventory control of a company that produces multiple types of toys.
- The goal is to determine the optimal quantities of each toy type to build and sell over a finite planning horizon, while meeting customer demand, inventory capacity, and production capacity constraints, and minimizing a given objective function (e.g., maximizing profit, minimizing costs, etc.).
- The model takes into account the minimum and maximum demand for each toy type in each period, the maximum number of toys that can be built in each period, the maximum inventory level, and the production cost and maximum inventory level for each type of toy.

- The decision variables of the model are the quantities of toys to make and sell in each period, as well as the end-of-period inventory levels for each type of toy.

FORMULATION:

Variables:

- Production[t][p]: the amount of toy type t produced in period p
- Sales[t][p]: the amount of toy type t sold in period p
- InStockAtEndOfPeriod[t][p]: the amount of toy type t in stock at the end of period p

Parameters:

- ToyTypes: a set of toy types
- NbPeriods: the number of periods
- MaxProductionPerPeriod[p]: the maximum production in period p
- MinDemand[t][p]: the minimum demand for toy type t in period p
- MaxDemand[t][p]: the maximum demand for toy type t in period p
- Toys[t]: a tuple representing a toy type t, with materials being the set of materials needed to produce the toy, productionCost being the cost of producing one unit of the toy, and maxInventory being the maximum inventory of the toy
- TotalProduction[t]: the total production for toy type t
- MaxInventory: the maximum inventory for all toy types

Objective:

- Minimize the total production cost:

  minimize sum (t in ToyTypes, p in Periods) Production[t][p] * Toys[t].productionCost

Constraints:

- Inventory capacity constraint: for each period p, the sum of the inventory for all toy types at the end of period p cannot exceed MaxInventory:

  forall(p in Periods) sum(t in ToyTypes) InStockAtEndOfPeriod[t][p] <= MaxInventory

- Maximum demand constraint: for each toy type t and period p, the amount of toy type t sold in period p cannot exceed MaxDemand[t][p]:

  forall(t in ToyTypes, p in Periods) Sales[t][p] <= MaxDemand[t][p]

- Minimum demand constraint: for each toy type t and period p, the amount of toy type t sold in period p must be at least MinDemand[t][p]:

  forall(t in ToyTypes, p in Periods) Sales[t][p] >= MinDemand[t][p]

- Maximum inventory constraint: for each toy type t and period p, the inventory of toy type t at the end of period p cannot exceed Toys[t].maxInventory:

  forall(t in ToyTypes, p in Periods) InStockAtEndOfPeriod[t][p] <= Toys[t].maxInventory

- Production capacity constraint: for each toy type t, the total production for toy type t must equal TotalProduction[t]:

  forall(t in ToyTypes) sum(p in Periods) Production[t][p] == TotalProduction[t]

- Initial inventory constraint: for each toy type t, the initial inventory at the start of period 1 must equal the sum of the sales in period 1 and the inventory at the end of period 1:

  forall(t in ToyTypes) Production[t][1] == Sales[t][1] + InStockAtEndOfPeriod[t][1]

- Inventory balance constraint: for each toy type t and period p greater than 1, the inventory at the end of period p-1 plus the production in period p must equal the sales in period p plus the inventory at the end of period p:

  forall(t in ToyTypes, p in 2..NbPeriods) InStockAtEndOfPeriod

PROGRAM CODE:

MOD FILE :

1. Defining data types:

```
{string} ToyTypes = ...;//Defining string for types of toys
{string} ComponentTypes = ...;//Defining string for types of components used to
make toys
int NbPeriods = ...;
range Periods = 1..NbPeriods;//Defining the range for number of periods
int MaxBuildPerPeriod[Periods] = ...; // Maximum production per period
int MaxDemand[ToyTypes][Periods] = ...; // Maximum demand per toy type per
period
int MinDemand[ToyTypes][Periods] = ...; // Minimum demand per toy type per
period
tuple toysToBuild { // Tuple representing a toy to produce
{string} components;
int price;
int maxInventory;
}
toysToBuild Toys[ToyTypes] = ...;
float TotalBuild[ToyTypes] = ...; // Total production of each toy type
int MaxInventory = 25;
```

2. Introducing decision variables:

```
dvar float+ Build[ToyTypes][Periods]; // Decision variables for production
dvar float+ Sell[ToyTypes][Periods]; // Decision variables for sales
dvar float+ InStockAtEndOfPeriod[ToyTypes][Periods]; // Decision variables for
inventory
```

3. Constraints:

```
subject to {
// Inventory capacity constraint
forall(p in Periods)
ctInventoryCapacity:
sum(t in ToyTypes) InStockAtEndOfPeriod[t][p] <= MaxInventory;

// Maximum demand constraint
forall(t in ToyTypes, p in Periods)
ctUnderMaxDemand: Sell[t][p] <= MaxDemand[t][p];

// Maximum inventory constraint
forall(t in ToyTypes, p in Periods)
ctToyTypeInventoryCapacity:
InStockAtEndOfPeriod[t][p] <= Toys[t].maxInventory;

// Minimum demand constraint
forall(t in ToyTypes, p in Periods)
ctOverMinDemand: Sell[t][p] >= MinDemand[t][p];

// Initial inventory constraint
forall(t in ToyTypes)
Build[t][1] == Sell[t][1] + InStockAtEndOfPeriod[t][1];

// Production capacity constraint
forall(t in ToyTypes)
ctTotalToBuild:
sum(p in Periods) Build[t][p] == TotalBuild[t];

// Inventory balance constraint
forall(t in ToyTypes, p in 2..NbPeriods)
ctInventoryBalance:
InStockAtEndOfPeriod[t][p-1] + Build[t][p] ==
Sell[t][p] + InStockAtEndOfPeriod[t][p];
}
```

DAT FILE :

```
ToyTypes={"Doll", "Car", "Train"};
ComponentTypes = {"Body", "Wheels", "Engine", "Battery"};
NbPeriods = 4;
MaxBuildPerPeriod = [5, 6, 7, 8];
MaxDemand = [[3, 5, 3, 5], [5, 3, 5, 3], [3, 3, 3, 3]];
MinDemand = [[1, 2, 1, 2], [2, 1, 2, 1], [1, 1, 1, 1]];
```

```
Toys = [

<{"Body", "Wheels", "Battery"}, 20, 15>,
<{"Body", "Wheels", "Engine"}, 25, 10>,
<{"Body", "Wheels", "Engine", "Battery"}, 30, 12>
];

TotalBuild = [30, 40, 20];
```

OUTPUT:

```
// solution (feasible relaxed sum of infeasibilities) with objective 21
InStockAtEndOfPeriod = [[0 0 0 14] [0 0 0 3] [0 0 0 8]];
Sell = [[3 5 3 5] [26 3 5 3] [3 3 3 3]];
Build = [[3 5 3 19] [26 3 5 6] [3 3 3 11]];
```

COMMENTS:

In the presentation, we have received a feedback that our problem statement was very basic. So, we improved on the production planning problem and are submitting a report for the multi-period production planning problem.