17. Inventory Management is Important! Optimization Case Study

LIU, Liangjie
Student ID: 21094937

MSc in DDM
HKUST

May 20, 2025

Problem Recap

- **Company:** Furniture Base
- **Issue:** Limited warehouse space; need to optimize which kitchen sets and styles to stock
- **Goal:** Maximize the number of kitchen sets that can be assembled from warehouse inventory
- **Constraints:** Storage limits on each component category (tiles, wallpaper, cabinets, etc.)

Kitchen Set Composition

- Each kitchen set is composed of 8 features:
  1. Tile (T1–T4)
  2. Wallpaper (W1–W4)
  3. Light Fixture (L1–L4)
  4. Cabinets (C1–C4)
  5. Countertops (O1–O4)
  6. Dishwasher (D1–D2) – optional
  7. Sink (S1–S4)
  8. Oven (V1–V4)
- 20 predefined kitchen sets with various combinations

Mathematical Model

Decision Variables:

- $y_i \in \{0, 1\}$: whether to include kitchen set i
- $z_s \in \{0, 1\}$: whether to stock style s

Objective: Maximize total number of kitchen sets:

$$\max \sum_{i=1}^{20} y_i$$

Constraints:

- Each part in a selected set must be available in inventory:

$$y_i \leq z_s \quad \forall s \in \text{parts of set } i$$

- Style capacity constraints, e.g.:

$$\sum_{s \in \text{Tiles}} z_s \leq 2, \qquad \sum_{s \in \text{Wallpapers}} z_s \leq 2, \ldots$$

Implementation: Python + PuLP

- Used `pulp` to build and solve Binary Integer Programming model
- Binary variables: y_i for sets, z_s for style availability

Sample Code: Objective and Constraints

```
model += pulp.lpSum(y[i] for i in sets)
# Objective: max set count

for i in sets:
    for part in sets[i]:
        if part is not None:
            model += y[i] <= z[part]
            # Set uses only stocked parts

model += pulp.lpSum(z[s] for s in tiles) <= 2
model += pulp.lpSum(z[s] for s in ovens) <= 3
```

Solution A: Base Capacity Constraints

Constraints:

- Max 2 styles each for tiles, wallpaper, lights, cabinets, sinks
- Max 3 styles of countertops
- Max 4 total styles among dishwashers and ovens

Results:

- Optimal kitchen sets assembled: **4**
- Selected Sets: `Set 4, 13, 14, 16`

Base Case: Styles to Stock

Selected Styles:

C1, C3, D1, L3, L4, O1, O2, O3,

S1, S2, T3, T4, V1, V3, V4, W3, W4

Insight:

- Emphasis on compact combinations using minimal overlapping features
- Dishwasher-optional sets included due to lower constraints

Solution C: Dishwasher+Oven Expansion

New Constraint:

- Max 2 styles each for tiles, wallpaper, lights, cabinets, sinks
- Max 3 styles of countertops
- Both D1 and D2 required
- Max 3 oven styles

Results:

- Optimal kitchen sets assembled: **5**
- Selected Sets: `Set 4, 8, 11, 15, 20`

Part C: Styles to Stock

Styles in Warehouse:

```
C1, C3, D1, D2, L1, L3, O1, O2, O3,

S1, S3, T2, T3, V1, V3, V4, W1, W3
```

Insight:

- Adding oven space increases viable set diversity
- Higher overlap between sets helps utilize style limits

Solution D: Full Expansion

New Limits:

- Max 2 styles each for tiles, wallpaper
- Max 3 styles for: lights, cabinets, ovens
- All styles of: dishwashers, sinks, countertops

Results:

- Optimal kitchen sets assembled: **8**
- Selected Sets: `4, 5, 13, 14, 15, 16, 19, 20`

Part D: Styles to Stock

Warehouse Inventory:

```
C1, C3, C4, D1, D2, L1, L3, L4, O1--O4,

S1--S4, T3, T4, V1, V3, V4, W3, W4
```

Impact:

- 100% increase in number of combinations from base case

Summary

- Binary Integer programming model helps optimally allocate warehouse inventory
- Significant improvement seen when increasing available storage
- More style overlap allows more flexible set selections
- Code-based implementation ensures reproducibility and scalability