Product Allocation and Functional Area Sizing

- Product Allocation and Functional Area Sizing (Min. Cost) (5pts)
- Block Layout Design (6 Departments) (5pts)
  - (Optional) Robust Block Layout Design (All departments) (Bonus 2.5pts)

In [2]:
```python
import pandas as pd
import math
import pulp
import gurobipy as gp
from gurobipy import GRB
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math
from collections import defaultdict
```

In [2]:
```python
"""
In this part of the code we import all the data into dataframes
"""

# Define product data
requirements = {
    "Product": ["Product 1", "Product 2", "Product 3", "Product 4", "Product 5", "Product 6"],
    "Annual demand (units)": [10000, 15000, 25000, 2000, 1500, 95000],
    "Order cost ($)": [50, 50, 50, 50, 50, 150],
    "Price/unit load ($)": [500, 650, 350, 250, 225, 150],
    "Space required (m²)": [10, 15, 25, 10, 12, 13],
    "Reserve dwell percentage (%)": [0, 0, 0.20, 0, 0, 1.00],
    "Yearly carrying cost rate (%)": [0.10, 0.10, 0.10, 0.10, 0.10, 0.10]
}

# Define flow data
flow_cost = {
    "Flow/Product": ["Flow 1 (CD)", "Flow 2 (R)", "Flow 3 (RF)", "Flow 4 (F)"],
    "Product 1": [0.0707, 0.0849, 0.1061, 0.0778],
    "Product 2": [0.0203, 0.2023, 0.2023, 0.2023],
    "Product 3": [0.0267, 0.0420, 0.0054, 0.0481],
    "Product 4": [0.3354, 0.5590, 1.0062, 0.0671],
    "Product 5": [0.4083, 0.6804, 1.2248, 0.8165],
    "Product 6": [0.0726, 0.0871, 0.1088, 0.0798]
}


# Define flow data (integer version)
yearly_cost = {
    "Flow/Product": ["Flow 1 (CD)", "Flow 2 (R)", "Flow 3 (RF)", "Flow 4 (F)"],
    "Product 1": [20, 5, 10, 15],
    "Product 2": [15, 5, 10, 10],
    "Product 3": [4, 20, 1, 9],
    "Product 4": [5, 4, 5, 1],
    "Product 5": [15, 25, 45, 30],
    "Product 6": [20, 5, 10, 15]
}

area_bounds = {
    "Functional Area": ["Cross-docking", "Reserve", "Forward"],
    "Lower bound (m²)": [0, 35000, 35000],
    "Upper bound (m²)": [15000, 75000, 75000]
}

levels = {
    "Functional Area": ["Cross-docking", "Reserve", "Forward"],
    "#Levels": [1, 1, 1]
}

area_bounds = pd.DataFrame(area_bounds)
levels = pd.DataFrame(levels)
yearly_cost = pd.DataFrame(yearly_cost)
```

```python
flow_cost = pd.DataFrame(flow_cost)
requirements  = pd.DataFrame(requirements)
```

In [3]:
```python
"""
In this part of the code we calculate the EOQ and average dwell time for each product usin the formulas from the slides
"""


eoq = []
avg_dwell = []
for i, row in requirements.iterrows():  # iterate over rows
    demand = row["Annual demand (units)"]
    order_cost = row["Order cost ($)"]
    price = row["Price/unit load ($)"]
    carrying_rate = row["Yearly carrying cost rate (%)"]

    eoq_val = math.sqrt((2 * demand * order_cost) / (price * carrying_rate))
    eoq.append(eoq_val)
    avg_dwell_time = eoq_val /(2 * demand)
    avg_dwell.append(avg_dwell_time)

# Add EOQ as a new column to the DataFrame
requirements["EOQ"] = eoq
requirements["Avg dwell time"] = avg_dwell

print(requirements[["Product", "EOQ", "Avg dwell time"]])
```

```
     Product          EOQ  Avg dwell time
0  Product 1   141.421356        0.007071
1  Product 2   151.910905        0.005064
2  Product 3   267.261242        0.005345
3  Product 4    89.442719        0.022361
4  Product 5    81.649658        0.027217
5  Product 6  1378.404875        0.007255
```

Mathematical formulation

Sets

- $I \in \{1,2,3,4,5,6\}$ are products which need to be stored in the warehouse.
- $J \in \{1,2,3,4\}$ are the different flows going trough the warehouse

Parameters

- S^{total} - Total availibale stoage space $(100.000m^2)$
- S_i - Space required for storing a unit of product i
- z^{CD} - Levels of space available in the vertical dimention of functional area Cross Docking
- z^F - Levels of space available in the vertical dimention of functional area Forward
- z^R - Levels of space available in the vertical dimention of functional area Reserve
- LL_{CD} - Lower storage space limit in the in cross docking $(0m^2)$
- UL_{CD} - Upper storage space limit in the in cross docking $(15000m^2)$
- LL_F - Lower storage space limit in the in forward $(35000m^2)$
- UL_F - Upper storage space limit in the in forward $(75000m^2)$
- LL_R - Lower storage space limit in the in reserve $(35000m^2)$
- UL_R - Upper storage space limit in the in reserve $(75000m^2)$
- C_{ij}^{handle} - Cost of handeling a unit load of product i and material flow j
- C_{ij}^{store} - Cost of storing a unit load of product i and material flow j
- ρ_i^R - Average percentage of time a unit load of product i spends in the reserve area if product is assigned to material flow 3

Decision variables

- $x_{ij} = \begin{cases} 1, & \text{if product i is assigned to flow j} \\ 0, & @ \end{cases}$
- w^{CD} Propotion of available space assigned to the crossing dock function area

- w^F Propotion of available space assigned to the forward function area
- w^R Propotion of available space assigned to the reserve function area

Objective function

$$min \sum_{i=1}^{6}\sum_{j=1}^{4}(2C_{ij}^{handle})D_i x_{ij} + \sum_{i=1}^{6}\sum_{j=1}^{4}\frac{Q_i}{2}C_{ij}^{store}x_{ij}$$

Same as in the lecture slides of this course, we assume for all flow only 2 transactions. Therefore, we only have the coefficient 2 here.

Constraints

Since we assign each product to one flow this will add up to one for all the flows combined combined with the fact that x is binary this function will work.

$$\sum_{j}^{4} x_{ij} = 1 \quad \forall i \in I$$

Since the cross docking only happens in flow 1:

$$\sum_{i=1}^{6}\frac{Q_i}{2}S_i x_{i1} \leq w^{CD}(z^{CD}S^{total})$$

Since the reserve only goes trough flow 2 and 3 the sum of these two sould be less than propotion of availbe space times the total space (vertial and in floor area).

$$\sum_{i=1}^{6}\frac{Q_i}{2}S_i x_{i2} + \sum_{i=1}^{6}\frac{Q_i}{2}\rho_i^R S_i x_{i3} \leq w^R(z^R S^{total})$$

Since the forward only goes trough flow 3 and 4 the sum of these two sould be less than propotion of availbe space times the total space (vertial and in floor area).

$$\sum_{i=1}^{6}\frac{Q_i}{2}(1-\rho_i^R)S_i x_{i3} + \sum_{i=1}^{6}\frac{Q_i}{2}S_i x_{i4} \leq w^F(z^F S^{total})$$

A 100% of the space should be allocated

$$w^{CD} + w^R + w^F = 1$$

The lower and upper limits given should be enforced

$$LL_{CD} \leq w^{CD}(z^{CD}S^{total}) \leq UL_{CD} LL_R \leq w^R(z^R S^{total}) \leq UL_R LL_F \leq w^F(z^F S^{total}) \leq UL_F$$

Proportion should be smaller than the avaible space

$$w^{CD}, w^R, w^F \geq 0$$

In [4]:

```
# Sets
products = requirements["Product"].tolist()  # I
flows = flow_cost["Flow/Product"].tolist()  # J

# Parameters
S_total = 100000
S = requirements.set_index("Product")["Space required (m²)"].to_dict()  # space per product
Q = dict(zip(requirements["Product"], requirements["EOQ"]))  # EOQ per product
C_handle = flow_cost.set_index("Flow/Product").T.to_dict()  # handling cost per product per flow
C_store = yearly_cost.set_index("Flow/Product").T.to_dict()  # storage cost per product per flow
rho_R = requirements.set_index("Product")["Reserve dwell percentage (%)"].to_dict()  # proportion to reserve

# Vertical levels
z_CD = levels.set_index("Functional Area").loc["Cross-docking", "#Levels"]
z_F = levels.set_index("Functional Area").loc["Forward", "#Levels"]
z_R = levels.set_index("Functional Area").loc["Reserve", "#Levels"]
```

```
# Limits
LL_CD, UL_CD = area_bounds.set_index("Functional Area").loc["Cross-docking", ["Lower bound (m²)", "Upper bou
LL_F, UL_F = area_bounds.set_index("Functional Area").loc["Forward", ["Lower bound (m²)", "Upper bound (m²)"
LL_R, UL_R = area_bounds.set_index("Functional Area").loc["Reserve", ["Lower bound (m²)", "Upper bound (m²)"

# Create model
model = pulp.LpProblem("Warehouse_Storage_Optimization", pulp.LpMinimize)

# Decision variables
x = pulp.LpVariable.dicts("x", [(i,j) for i in products for j in flows], cat='Binary')
w_CD = pulp.LpVariable("w_CD", lowBound=0)
w_F  = pulp.LpVariable("w_F", lowBound=0)
w_R  = pulp.LpVariable("w_R", lowBound=0)

# Objective function
total_cost = pulp.lpSum([
    (2 * C_handle[j][i] * Q[i] * x[i,j]) +
    (Q[i]/2 * C_store[j][i] * x[i,j])
    for i in products for j in flows
])

# Add to objective
model += total_cost
# Constraints

# Each product assigned to exactly one flow
for i in products:
    model += pulp.lpSum([x[i,j] for j in flows]) == 1

# Cross Docking (Flow 1)
model += pulp.lpSum([Q[i]/2 * S[i] * x[i,'Flow 1 (CD)'] for i in products]) <= w_CD * z_CD * S_total

# Reserve (Flow 2 and Flow 3)
model += pulp.lpSum([Q[i]/2 * S[i] * x[i,'Flow 2 (R)'] + Q[i]/2 * rho_R[i] * S[i] * x[i,'Flow 3 (RF)'] for i

# Forward (Flow 3 and Flow 4)
model += pulp.lpSum([Q[i]/2 * (1-rho_R[i]) * S[i] * x[i,'Flow 3 (RF)'] + Q[i]/2 * S[i] * x[i,'Flow 4 (F)'] fo

# 100% of the space allocated
model += w_CD + w_R + w_F == 1

# Enforce lower and upper limits
model += w_CD * z_CD * S_total >= LL_CD
model += w_CD * z_CD * S_total <= UL_CD
model += w_R * z_R * S_total >= LL_R
model += w_R * z_R * S_total <= UL_R
model += w_F * z_F * S_total >= LL_F
model += w_F * z_F * S_total <= UL_F

# Solve the model
model.solve()

# Output results
print("Product Allocations:")
for i in products:
    for j in flows:
        if x[i,j].value() == 1:
            print(f"{i}: {j}")
print()
print("Area Sizes:")
print(f"Cross Docking: {w_CD.value()*S_total}m²")
print(f"Reserve: {w_R.value()*S_total}m²")
print(f"Forward: {w_F.value()*S_total}m²")
print()
print("Total cost:", round(pulp.value(total_cost), 2))
```

```
Product Allocations:
Product 1: Flow 2 (R)
Product 2: Flow 2 (R)
Product 3: Flow 3 (RF)
Product 4: Flow 4 (F)
Product 5: Flow 1 (CD)
Product 6: Flow 2 (R)

Area Sizes:
Cross Docking: 15000.0m²
Reserve: 35000.0m²
Forward: 50000.0m²

Total cost: 5377.23
```

11.5 Warehouse Block Layout Design

In [8]:
```
# Example: access a value
#print(departments["Cross-Dock"])  # Output: 3520
departments_matrix = {
    "Inbound Dock": {
        "Inbound Dock": "-",
        "Receiving/Staging": "E",
        "QA & Technical Test": "U",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "U",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Receiving/Staging": {
        "Inbound Dock": "E",
        "Receiving/Staging": "-",
        "QA & Technical Test": "A",
        "Cross-Dock": "A",
        "Pallet Reserve Storage (Bulk)": "I",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "U",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "I",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "QA & Technical Test": {
        "Inbound Dock": "U",
        "Receiving/Staging": "A",
        "QA & Technical Test": "-",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "U",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "I",
        "Spare Parts & Accessories Cage": "U"
    },
    "Cross-Dock": {
        "Inbound Dock": "U",
        "Receiving/Staging": "A",
        "QA & Technical Test": "U",
        "Cross-Dock": "-",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "U",
```

```
        "Outbound Staging": "A",
        "Shipping Dock": "A",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Pallet Reserve Storage (Bulk)": {
        "Inbound Dock": "U",
        "Receiving/Staging": "I",
        "QA & Technical Test": "U",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "-",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "E",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "O",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Oversize/Non-Standard Storage": {
        "Inbound Dock": "U",
        "Receiving/Staging": "U",
        "QA & Technical Test": "U",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "-",
        "Packing / Wrap / Banding": "I",
        "Outbound Staging": "U",

        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Packing / Wrap / Banding": {
        "Inbound Dock": "U",
        "Receiving/Staging": "U",
        "QA & Technical Test": "U",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "E",
        "Oversize/Non-Standard Storage": "I",
        "Packing / Wrap / Banding": "-",
        "Outbound Staging": "E",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "O",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "O",
        "Spare Parts & Accessories Cage": "U"
    },
    "Outbound Staging": {
        "Inbound Dock": "U",
        "Receiving/Staging": "U",
        "QA & Technical Test": "U",
        "Cross-Dock": "A",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "E",
        "Outbound Staging": "-",
        "Shipping Dock": "E",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Shipping Dock": {
        "Inbound Dock": "U",
        "Receiving/Staging": "U",
        "QA & Technical Test": "U",
        "Cross-Dock": "A",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
```

```
        "Packing / Wrap / Banding": "U",
        "Outbound Staging": "E",
        "Shipping Dock": "-",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Empty Pallets & Dunnage": {
        "Inbound Dock": "U",
        "Receiving/Staging": "I",
        "QA & Technical Test": "U",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "O",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "-",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Maintenance & Battery Charge": {
        "Inbound Dock": "U",
        "Receiving/Staging": "U",
        "QA & Technical Test": "U",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "O",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "U",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "-",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "U"
    },
    "Returns & WEEE": {
        "Inbound Dock": "U",
        "Receiving/Staging": "U",
        "QA & Technical Test": "I",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "O",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "-",
        "Spare Parts & Accessories Cage": "U"
    },
    "Spare Parts & Accessories Cage": {
        "Inbound Dock": "U",
        "Receiving/Staging": "U",
        "QA & Technical Test": "U",
        "Cross-Dock": "U",
        "Pallet Reserve Storage (Bulk)": "U",
        "Oversize/Non-Standard Storage": "U",
        "Packing / Wrap / Banding": "U",
        "Outbound Staging": "U",
        "Shipping Dock": "U",
        "Empty Pallets & Dunnage": "U",
        "Maintenance & Battery Charge": "U",
        "Returns & WEEE": "U",
        "Spare Parts & Accessories Cage": "-"
    }
}

# Define the mapping of letters to numeric values
letter_to_number = {
    "E": 4,
    "A": 3,
    "I": 2,
```

```python
    "O": 1,
    "U": 0,
    "-": 0
}
# Selected six departments
def load_data():
    selected_depts = [
        "Inbound Dock",
        "Outbound Staging",
        "Packing / Wrap / Banding",
        "Pallet Reserve Storage (Bulk)",
        "Receiving/Staging",
        "Shipping Dock"
    ]
    return selected_depts


def load_all_data():
    selected_depts = [
        "Inbound Dock",
        "Outbound Staging",
        "Packing / Wrap / Banding",
        "Pallet Reserve Storage (Bulk)",
        "Receiving/Staging",
        "Shipping Dock",
        "Cross-Dock",
        "Empty Pallets & Dunnage",
        "Maintenance & Battery Charge",
        "Oversize/Non-Standard Storage",
        "QA & Technical Test",
        "Returns & WEEE",
        "Spare Parts & Accessories Cage"
    ]
    return selected_depts
departments = {
    "Cross-Dock": 3520,
    "Empty Pallets & Dunnage": 880,
    "Inbound Dock": 2640,
    "Maintenance & Battery Charge": 1320,
    "Outbound Staging": 8800,
    "Oversize/Non-Standard Storage": 2640,
    "Packing / Wrap / Banding": 3520,
    "Pallet Reserve Storage (Bulk)": 46340,
    "QA & Technical Test": 1760,
    "Receiving/Staging": 5280,
    "Returns & WEEE": 2640,
    "Shipping Dock": 3520,
    "Spare Parts & Accessories Cage": 440
}

# Example: access a value
#print(departments_matrix["Inbound Dock"]["Receiving/Staging"])  # Output: E
```

Translation from letter to numeric

The convert_letters_to_numbers function translates qualitative relationship scores—represented by letters (E, A, I, O, U, -)—into quantitative values using a predefined mapping, where each letter corresponds to a numeric priority. These values are then squared to amplify the importance of higher-priority relationships, ensuring that critical connections (like "E") have a significantly greater impact on the optimization process this will make the simulation run faster.

In [9]:
```python
# Conversion based on SLP legend
# Function to convert nested dictionary values
def convert_letters_to_numbers(matrix, mapping):
    numeric_matrix = {}
    for dept_from, relations in matrix.items():
        numeric_matrix[dept_from] = {}
        for dept_to, letter in relations.items():
            numeric_matrix[dept_from][dept_to] = mapping.get(letter, None)  # None if unknown
    return numeric_matrix
# Apply the conversion
numeric_departments_matrix = convert_letters_to_numbers(departments_matrix, letter_to_number)
# Example: check numeric value
```

```
print(numeric_departments_matrix["Inbound Dock"]["Receiving/Staging"])  # Output: 4
print(numeric_departments_matrix["Inbound Dock"]["Inbound Dock"])      # Output: 0
```

```
4
0
```

In [10]:
```
# Conversion based on squared SLP legend
# Function to convert nested dictionary values
def convert_letters_to_numbers(matrix, mapping):
    numeric_matrix = {}
    for dept_from, relations in matrix.items():
        numeric_matrix[dept_from] = {}
        for dept_to, letter in relations.items():
            val = mapping.get(letter, None)
            numeric_matrix[dept_from][dept_to] = val**2 if val is not None else None
    return numeric_matrix

numeric_departments_matrix = convert_letters_to_numbers(departments_matrix, letter_to_number)

print(numeric_departments_matrix["Inbound Dock"]["Receiving/Staging"])  # Output: 16
print(numeric_departments_matrix["Inbound Dock"]["Inbound Dock"])      # Output: 0
```

```
16
0
```

Mathematical formulation

We consider the following 6 departments

- Inbound Dock
- Receiving/Staging
- Pallet Reserve Storage (Bulk)
- Packing / Wrap / Banding
- Outbound Staging (Join areas for 2-Man Delivery and Parcel)
- Shipping Dock

Sets

- $I \in \{1, 2, 3, 4, 5, 6\}$ are the 6 departments
- $J \in \{1, 2, 3, 4, 5, 6\}$ are the 6 departments

Parameters

- m - Number of departments
- f_{ij} - Flow from department i to department j
- c_{ij} - Cost of moving a unit load one distance unit from department i to department j
- B_x - Building length (measured along the x-coordinate)
- B_y - Building width (measured along the y-coordinate)
- A_i - Area of department i
- L_i^{LB} - Lower limit on the length of department i
- L_i^{UB} - Upper limit on the length of department i
- W_i^{LB} - Lower limit on the width of department i
- W_i^{UB} - Upper limit on the width of department i
- M - Large number

Variables

- α_i: x-coordinate of the centroid of department i
- β_i: y-coordinate of the centroid of department i
- x_i^{left}: x-coordinate of the left (or west) side of department i
- x_i^{right}: x-coordinate of the right (or east) side of department i
- y_i^{bottom}: y-coordinate of the bottom (or south) side of department i
- y_i^{top}: y-coordinate of the top (or north) side of department i

- z_{ij}^x: 1 if department i is strictly to the east of department j; 0 otherwise
- z_{ij}^y: 1 if department i is strictly to the north of department j; 0 otherwise

Objective function

$$\min \sum_{i=1}^{6} \sum_{j=1}^{6} f_{ij} c_{ij} (|\alpha_i - \alpha_j| + |\beta_i - \beta_j|)$$

Constraints

The length and width of each department must remain within the specified bounds:

Length Constraints:

$$L_i^{\mathrm{LB}} \leq (x_i^{\mathrm{right}} - x_i^{\mathrm{left}}) \leq L_i^{\mathrm{UB}} \quad \forall i \in I$$

Width Constraints:

$$W_i^{\mathrm{LB}} \leq (y_i^{\mathrm{top}} - y_i^{\mathrm{bottom}}) \leq W_i^{\mathrm{UB}} \quad \forall i \in I$$

Area Constraints:

$$(x_i^{\mathrm{right}} - x_i^{\mathrm{left}})(y_i^{\mathrm{top}} - y_i^{\mathrm{bottom}}) = A_i \quad \forall i \in I$$

X-Direction Boundaries:

$$0 \leq x_i^{\mathrm{left}} \leq x_i^{\mathrm{right}} \leq B_x \quad \forall i \in I$$

Y-Direction Boundaries:

$$0 \leq y_i^{\mathrm{bottom}} \leq y_i^{\mathrm{top}} \leq B_y \quad \forall i \in I$$

X-Coordinate Centroid:

$$\alpha_i = 0.5 x_i^{\mathrm{left}} + 0.5 x_i^{\mathrm{right}} \quad \forall i \in I$$

Y-Coordinate Centroid:

$$\beta_i = 0.5 y_i^{\mathrm{bottom}} + 0.5 y_i^{\mathrm{top}} \quad \forall i \in I$$

East-West Separation:

$$x_j^{\mathrm{right}} \leq x_i^{\mathrm{left}} + M(1 - z_{ij}^x) \quad \forall i, j, i \neq j$$

North-South Separation:

$$y_j^{\mathrm{top}} \leq y_i^{\mathrm{bottom}} + M(1 - z_{ij}^y) \quad \forall i, j, i \neq j$$

Mutual Exclusion:

$$z_{ij}^x + z_{ji}^x + z_{ij}^y + z_{ji}^y \geq 1 \quad \forall i, j, i \leq j$$

Continuous Variable Non-negativity:

$$\alpha_i, \beta_i \geq 0 \quad \forall i$$

Boundary Variable Non-negativity:

$$x_i^{\mathrm{left}}, x_i^{\mathrm{right}}, y_i^{\mathrm{bottom}}, y_i^{\mathrm{top}} \geq 0 \quad \forall i$$

Binary Variable Declaration:

$$z_{ij}^x, z_{ij}^y \in \{0, 1\} \quad \forall i, j, i \neq j$$

Absolute Difference Constraints:

$$d\alpha_{ij} \geq \alpha_i - \alpha_j \quad \forall i, j$$

$$d\alpha_{ij} \geq -\alpha_i + \alpha_j \quad \forall i, j$$

$$d\beta_{ij} \geq \beta_i - \beta_j \quad \forall i, j$$

$$d\beta_{ij} \geq -\beta_i + \beta_j \quad \forall i, j$$

Aspect ratio:

I-shaped layout

- Inbound dock at one side of the I:

$$x^{\text{left}}_{\text{Inbound Dock}} = 0$$

- Outbound dock at the other side:

$$x^{\text{right}}_{\text{Outbound Staging}} = B_{\text{width}}$$

- Aspect ratio constraints:

$$B_{\text{width}} \leq I \cdot B_{\text{height}}, \quad B_{\text{height}} \leq I \cdot B_{\text{width}}$$

L-shaped layout

- Inbound dock at bottom-left corner:

$$x^{\text{left}}_{\text{Inbound Dock}} = 0$$

- Outbound dock at top-right corner:

$$x^{\text{right}}_{\text{Outbound Staging}} = B_{\text{width}}$$

- Phantom department at bottom wall:

$$x^{\text{right}}_{\text{Phantom}} - x^{\text{left}}_{\text{Phantom}} \geq \frac{1}{4} B_{\text{width}}, \quad y^{\text{top}}_{\text{Phantom}} - y^{\text{bottom}}_{\text{Phantom}} \geq \frac{1}{4} B_{\text{height}}$$

$$x^{\text{left}}_{\text{Phantom}} = 0, \quad y^{\text{bottom}}_{\text{Phantom}} = 0$$

U-shaped layout

- Inbound dock at bottom-left corner:

$$x^{\text{left}}_{\text{Inbound Dock}} = 0, \quad y^{\text{bottom}}_{\text{Inbound Dock}} = 0$$

- Outbound dock at top-left corner:

$$x^{\text{left}}_{\text{Outbound Staging}} = 0, \quad y^{\text{top}}_{\text{Outbound Staging}} = B_{\text{height}}$$

- Phantom department at left wall:

$$x^{\text{left}}_{\text{Phantom}} = 0, \quad x^{\text{right}}_{\text{Phantom}} + 1 \leq B_{\text{width}}$$

- Minimum dimensions for Phantom department to create a propper U shape:

$$x^{\text{right}}_{\text{Phantom}} - x^{\text{left}}_{\text{Phantom}} \geq \frac{1}{4} B_{\text{width}}, \quad y^{\text{top}}_{\text{Phantom}} - y^{\text{bottom}}_{\text{Phantom}} \geq \frac{1}{4} B_{\text{height}}$$

Code

Side Notes

Coverting Qualitative SLP

We converted the qualitative SLP to a quanitive one by using the SLP legend. In this case it was possible as 'X' was not given in the SLP adjacent matrix and we therefore, also do not make use of negative numbers. We tied it with the scores in the SLP legend and also by squaring the SLP legend scores. When squaring the values, the running time get be decreased.

Outbound Staging

For the outbound strategy we where suppose to ioin areas for 2-Man Delivery and Parcel. We have done this by adding their areas together. Additionally, for the interaction we compared if they have for all departments the same value. As this was the case, we used these values for a department called 'Outbound Staging' and took out the departments for 2-Man Delivery and Parcel.

Aspect Ratio

We make use of an aspect ratio, for giving the warehouse certain proportions. 'R' is used for the proportion of the departments and 'I' is used for the general shape of the I-shaped warehouse.

Clearance

As we limit our warehouse size to the sum of the area needed by each deaprtment, we do not make use of clearance. Otherwise our model would be infeasable as a larger area would be needed.

Phantom Department

In order to create L- and U- shaped warehouses, we introduced a phantom warehouse. For this department we added a consraint which defined on which wall (s) it should be placed to reach the wanted shape. The exact size of the phantom warehouse then depends on width and the height of the total warehouse.

Costs

As no costs have been specified in the assignment describtion, we chose to set c to 1. Therefore, we left it out in our objective unction in the gurobi model below.

In [13]:
```
selected_depts = load_data()

# ----------------------------
# USER PARAMETERS / INPUTS
# ----------------------------
# Department areas (sq_m) - scaled down by factor of 10 for more reasonable dimensions
L_LB = {d: 1.0 for d in selected_depts}
W_LB = {d: 1.0 for d in selected_depts}
R = 3  # Aspect ratio tolerance (1.2 means 1:1.2 to 1.2:1)
I = 4
BIG_M = 10000.0  # Big-M for separation constraints

# Calculate total area
total_area = sum(departments[d] for d in selected_depts)


def solve_layout(layout_type ,total_area=total_area, selected_depts=selected_depts, departments=departments,

    model = gp.Model(f"warehouse_layout_{layout_type}")
    model.Params.NonConvex = 2

    # Variables
    alpha, beta = {}, {}
    x_left, x_right, y_bottom, y_top = {}, {}, {}, {}
    width, height = {}, {}
    d_alpha = {}
    d_beta = {}
    z_x = {}
    z_y = {}
    B_width = model.addVar(lb=0, name="B_width")   # small lb to avoid zero
    B_height = model.addVar(lb=0, name="B_height")

    if layout_type == "L-shaped" or layout_type == "U-shaped":
        selected_depts.append("Phantom")
        departments["Phantom"] = total_area/3
        numeric_departments_matrix["Phantom"] = {d: 0 for d in selected_depts}
        for d in selected_depts:
            numeric_departments_matrix[d]["Phantom"] = 0

    for i in selected_depts:
        x_left[i] = model.addVar(lb=0, name=f"x_left_{i}")
        y_bottom[i] = model.addVar(lb=0, name=f"y_bottom_{i}")
        width[i] = model.addVar(lb=0, name=f"width_{i}")
        height[i] = model.addVar(lb=0, name=f"height_{i}")

        # Derived boundaries
```

```python
        x_right[i] = model.addVar(lb=0, name=f"x_right_{i}")
        y_top[i] = model.addVar(lb=0, name=f"y_top_{i}")
        alpha[i] = model.addVar(name=f"alpha_{i}")
        beta[i] = model.addVar(name=f"beta_{i}")
        # ensure nested dict entries exist
        d_alpha.setdefault(i, {})
        d_beta.setdefault(i, {})
        z_x.setdefault(i, {})
        z_y.setdefault(i, {})
        for j in selected_depts:
            print(i, j)
            d_alpha[i][j] = model.addVar(lb=0, name=f"d_alpha_{i}_{j}")
            d_beta[i][j] = model.addVar(lb=0, name=f"d_beta_{i}_{j}")
            z_x[i][j] = model.addVar(vtype=GRB.BINARY, name=f"z_x_{i}_{j}")
            z_y[i][j] = model.addVar(vtype=GRB.BINARY, name=f"z_y_{i}_{j}")

    model.update()

    # Placement, dimension & area constraints
    for i in selected_depts:
        # Within building
        model.addConstr(alpha[i] >= 0, name=f"alpha_pos_{i}")
        model.addConstr(beta[i] >= 0, name=f"beta_pos_{i}")
        model.addConstr(x_right[i] >= 0, name=f"width_lb_{i}")
        model.addConstr(x_left[i] >= 0, name=f"x_left_pos_{i}")
        model.addConstr(y_top[i] >= 0, name=f"y_top_pos_{i}")
        model.addConstr(y_bottom[i] >= 0, name=f"y_bottom_pos_{i}")
        model.addConstr(x_right[i] <= B_width, name=f"x_right_pos_{i}")
        model.addConstr(y_top[i] <= B_height, name=f"y_top_pos_{i}")

        # Area (bilinear but with positive factors only)
        area = departments[i]

        if i == "Phantom":
            model.addConstr((x_right[i] - x_left[i]) * (y_top[i] - y_bottom[i]) >= 1 * area, name=f"area_lb_
        else:
            model.addConstr((x_right[i] - x_left[i]) * (y_top[i] - y_bottom[i]) == 1 * area, name=f"area_lb_


        # Area Constraint (upper bound)
        model.addConstr((x_right[i]-x_left[i])  <= R *(y_top[i]-y_bottom[i]), name=f"area_ub_{i}")
        model.addConstr((y_top[i]-y_bottom[i])  <= R *(x_right[i]-x_left[i]), name=f"area_ub_{i}")

        # Centroids
        model.addConstr(alpha[i] == 0.5 * (x_left[i] + x_right[i]), name=f"centroid_x_{i}")
        model.addConstr(beta[i] == 0.5 * (y_bottom[i] + y_top[i]), name=f"centroid_y_{i}")

        for j in selected_depts:
            if i <= j:
                model.addConstr(z_x[i][j] + z_x[j][i] + z_y[i][j] + z_y[j][i] >= 1, name=f"mutual_excl_{i}_{
            if i != j:
                model.addConstr(x_right[j] <= x_left[i] + BIG_M*(1 - z_x[i][j]), name=f"width_def_{i}_{j}")
                model.addConstr(y_top[j] <= y_bottom[i] + BIG_M*(1 - z_y[i][j]), name=f"height_def_{i}_{j}")
                # Absolute distance linearization
                model.addConstr(d_alpha[i][j] >=  alpha[i] - alpha[j])
                model.addConstr(d_alpha[i][j] >= -alpha[i] + alpha[j])
                model.addConstr(d_beta[i][j]  >=  beta[i]  - beta[j])
                model.addConstr(d_beta[i][j]  >= -beta[i]  + beta[j])

    #space specific constraints
    if layout_type == "I-shaped":
        # Inbound dock at bottom
        # Inbound dock must be at the left wall
        model.addConstr(x_left["Inbound Dock"] == 0, name="inbound_left_wall")

        # Outbound dock must be at the right wall
        model.addConstr(x_right["Outbound Staging"] == B_width, name="outbound_right_wall")
        model.addConstr(B_width <= I * B_height)
        model.addConstr(B_height <= I * B_width)
        #total_area = total_area * 1

    if layout_type == "L-shaped":
        # Inbound dock at bottom left corner
        model.addConstr(x_left["Inbound Dock"] == 0, name="inbound_left_wall")
        model.addConstr(y_top["Inbound Dock"] == B_height, name="inbound_top_wall")
```

```python
        # Outbound dock at top right corner
        model.addConstr(x_right["Outbound Staging"] == B_width, name="outbound_right_wall")
        model.addConstr(y_bottom["Outbound Staging"] == 0, name="outbound_bottom_wall")
        # phantom department at bottom wall
        model.addConstr((x_right["Phantom"] - x_left["Phantom"]) >= 1/4 * B_width, name="inbound_width")
        model.addConstr((y_top["Phantom"] - y_bottom["Phantom"]) >= 1/4 * B_height, name="inbound_height")
        model.addConstr(x_left["Phantom"] == 0, name="inbound_left_wall")
        model.addConstr(y_bottom["Phantom"] == 0, name="inbound_bottom_wall")
    if layout_type == "U-shaped":

         # Inbound dock at bottom left corner
        model.addConstr(y_bottom["Inbound Dock"] == 0, name="inbound_bottom_wall")
        model.addConstr(x_left["Inbound Dock"] == 0, name="inbound_left_wall")
        # Outbound dock at top left corner
        model.addConstr(y_top["Outbound Staging"] == B_height, name="outbound_top_wall")
        model.addConstr(x_left["Outbound Staging"] == 0, name="outbound_left_wall")
        # phantom department at left wall
        model.addConstr(x_left["Phantom"] == 0, name="phantom_left_wall")
        model.addConstr(B_width  >=  x_right["Phantom"]+1, name="phantom_right")
        model.addConstr((x_right["Phantom"] - x_left["Phantom"]) >= 1/4 * B_width, name="inbound_width")
        model.addConstr((y_top["Phantom"] - y_bottom["Phantom"]) >= 1/4 * B_height, name="inbound_height")



    total_area = sum(departments[d] for d in selected_depts)
    print(f"Total area for {layout_type}: {total_area}")
    model.addConstr(B_width * B_height == total_area, name="total_area")
    # Objective
    obj = gp.quicksum(
        numeric_departments_matrix[i][j] * (d_alpha[i][j] + d_beta[i][j])
        for i in selected_depts for j in selected_depts if i != j
    )
    model.setObjective(obj, GRB.MINIMIZE)

    model.optimize()

    # Plot results (unchanged except width/height already consistent)
    if model.status in (GRB.OPTIMAL, GRB.TIME_LIMIT, GRB.SUBOPTIMAL):
        print(f"\n--- {layout_type.upper()} LAYOUT ---")
        print(f"Building dimensions: {B_width.X:.1f} x {B_height.X:.1f} m")
        print(f"Total area: {B_width.X * B_height.X:.1f} sqm")
        print(f"Objective (total flow-distance): {model.ObjVal:.1f}")

        # ...existing code inside solve_layout, in the plotting section after creating fig, ax ...
        fig, ax = plt.subplots(figsize=(15,10))
        colors = plt.cm.get_cmap('tab20', len(selected_depts))

        # Warehouse (building) outline
        building_outline = patches.Rectangle(
            (0, 0),
            B_width.X,
            B_height.X,
            facecolor='none',
            edgecolor='red',
            linewidth=2,
            linestyle='--',
            label='Building'
        )
        ax.add_patch(building_outline)
        ax.text(B_width.X/2, B_height.X + 0.02*B_height.X,
                f"Warehouse: {B_width.X:.1f} x {B_height.X:.1f} m",
                ha='center', va='bottom', fontsize=12)

        for idx, i in enumerate(selected_depts):
            xl, xr = x_left[i].X, x_right[i].X
            yb, yt = y_bottom[i].X, y_top[i].X

            # Check if the department is Phantom
            if i == "Phantom":
                facecolor = 'white'
                label_text = ""  # no name
                show_marker = False
                show_area = False
            else:
                facecolor = colors(idx)
```

```python
                label_text = i
                show_marker = True
                show_area = True

            rect = patches.Rectangle((xl, yb), xr - xl, yt - yb,
                                    edgecolor='black', facecolor=facecolor, alpha=0.7)
            ax.add_patch(rect)

            # Add department name
            if label_text:
                ax.text((xl + xr)/2, (yb + yt)/2, label_text, ha='center', va='center',
                        fontsize=10, fontweight='bold',
                        bbox=dict(facecolor='white', alpha=0.7, edgecolor='none'))

            # Add marker if not Phantom
            if show_marker:
                ax.plot((xl + xr)/2, (yb + yt)/2, 'kx', markersize=8)

            # Add area text if not Phantom
            if show_area:
                ax.text(xl + 5, yt - 5, f"{(xr - xl)*(yt - yb):.0f} sqm", fontsize=8)

        ax.set_xlim(0, B_width.X)
        ax.set_ylim(0, B_height.X)
        ax.set_aspect('equal')
        ax.set_title(f"{layout_type} Layout\nTotal Flow-Distance: {model.ObjVal:.1f}", fontsize=14)
        ax.grid(True, linestyle='--', alpha=0.6)
        ax.set_xlabel("Width (m)")
        ax.set_ylabel("Height (m)")
        ax.legend(loc='upper right')
        plt.show()

    else:
        print(f"No feasible solution for {layout_type}, status: {model.status}")
```

In [14]:

```python
# Run for all layouts
for layout_type in [ "L-shaped"]:
    selected_depts = load_data()
    solve_layout(layout_type, total_area=total_area, selected_depts=selected_depts, departments=departments,
```

```
Set parameter Username
Set parameter LicenseID to value 2620689
Academic license - for non-commercial use only - expires 2026-02-11
Set parameter NonConvex to value 2
Inbound Dock Inbound Dock
Inbound Dock Outbound Staging
Inbound Dock Packing / Wrap / Banding
Inbound Dock Pallet Reserve Storage (Bulk)
Inbound Dock Receiving/Staging
Inbound Dock Shipping Dock
Inbound Dock Phantom
Outbound Staging Inbound Dock
Outbound Staging Outbound Staging
Outbound Staging Packing / Wrap / Banding
Outbound Staging Pallet Reserve Storage (Bulk)
Outbound Staging Receiving/Staging
Outbound Staging Shipping Dock
Outbound Staging Phantom
Packing / Wrap / Banding Inbound Dock
Packing / Wrap / Banding Outbound Staging
Packing / Wrap / Banding Packing / Wrap / Banding
Packing / Wrap / Banding Pallet Reserve Storage (Bulk)
Packing / Wrap / Banding Receiving/Staging
Packing / Wrap / Banding Shipping Dock
Packing / Wrap / Banding Phantom
Pallet Reserve Storage (Bulk) Inbound Dock
Pallet Reserve Storage (Bulk) Outbound Staging
Pallet Reserve Storage (Bulk) Packing / Wrap / Banding
Pallet Reserve Storage (Bulk) Pallet Reserve Storage (Bulk)
Pallet Reserve Storage (Bulk) Receiving/Staging
Pallet Reserve Storage (Bulk) Shipping Dock
Pallet Reserve Storage (Bulk) Phantom
Receiving/Staging Inbound Dock
Receiving/Staging Outbound Staging
Receiving/Staging Packing / Wrap / Banding
Receiving/Staging Pallet Reserve Storage (Bulk)
Receiving/Staging Receiving/Staging
Receiving/Staging Shipping Dock
Receiving/Staging Phantom
Shipping Dock Inbound Dock
Shipping Dock Outbound Staging
Shipping Dock Packing / Wrap / Banding
Shipping Dock Pallet Reserve Storage (Bulk)
Shipping Dock Receiving/Staging
Shipping Dock Shipping Dock
Shipping Dock Phantom
Phantom Inbound Dock
Phantom Outbound Staging
Phantom Packing / Wrap / Banding
Phantom Pallet Reserve Storage (Bulk)
Phantom Receiving/Staging
Phantom Shipping Dock
Phantom Phantom
Total area for L-shaped: 93466.66666666667
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (win64 - Windows 11.0 (26100.2))

CPU model: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Non-default parameters:
NonConvex  2

Optimize a model with 372 rows, 254 columns and 1036 nonzeros
Model fingerprint: 0x7f0b74e3
Model has 8 quadratic constraints
Variable types: 156 continuous, 98 integer (98 binary)
Coefficient statistics:
  Matrix range     [3e-01, 1e+04]
  QMatrix range    [1e+00, 1e+00]
  Objective range  [4e+00, 2e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
  QRHS range       [3e+03, 9e+04]
Presolve removed 223 rows and 140 columns
Presolve time: 0.00s
```

```
Presolved: 239 rows, 135 columns, 653 nonzeros
Presolved model has 1 quadratic constraint(s)
Presolved model has 21 bilinear constraint(s)
Warning: Model contains variables with very large bounds participating
         in product terms.
         Presolve was not able to compute smaller bounds for these variables.
         Consider bounding these variables or reformulating the model.


Solving non-convex MIQCP

Variable types: 62 continuous, 73 integer (73 binary)

Root relaxation: objective 3.735176e+01, 85 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   37.35176    0   38          -   37.35176      -     -    0s
     0     0 1222.89275    0   38          - 1222.89275      -     -    0s
     0     0 1222.89275    0   38          - 1222.89275      -     -    0s
     0     0 1350.29007    0   36          - 1350.29007      -     -    0s
     0     0 1350.29007    0   36          - 1350.29007      -     -    0s
     0     0 1638.53954    0   32          - 1638.53954      -     -    0s
     0     0 1638.53954    0   34          - 1638.53954      -     -    0s
     0     0 1639.17638    0   38          - 1639.17638      -     -    0s
     0     0 1639.17638    0   38          - 1639.17638      -     -    0s
     0     0 1639.17643    0   36          - 1639.17643      -     -    0s
     0     0 1639.17643    0   36          - 1639.17643      -     -    0s
     0     0 1773.37920    0   36          - 1773.37920      -     -    0s
     0     0 1782.12895    0   36          - 1782.12895      -     -    0s
     0     0 1951.34961    0   38          - 1951.34961      -     -    0s
     0     0 1965.80473    0   38          - 1965.80473      -     -    0s
     0     0 1977.01817    0   37          - 1977.01817      -     -    0s
     0     0 1977.01817    0   37          - 1977.01817      -     -    0s
     0     0 1984.25672    0   33          - 1984.25672      -     -    0s
     0     0 1985.12111    0   35          - 1985.12111      -     -    0s
     0     0 1993.90882    0   35          - 1993.90882      -     -    0s
     0     0 1994.01387    0   35          - 1994.01387      -     -    0s
     0     0 1994.28439    0   35          - 1994.28439      -     -    0s
     0     0 1994.28439    0   35          - 1994.28439      -     -    0s
     0     0 1994.38363    0   35          - 1994.38363      -     -    0s
     0     0 1994.43205    0   35          - 1994.43205      -     -    0s
     0     0 1994.43903    0   35          - 1994.43903      -     -    0s
     0     0 2029.56753    0   29          - 2029.56753      -     -    0s
     0     2 2029.56753    0   29          - 2029.56753      -     -    0s
 33901 15697 infeasible   53               - 8916.28977      -   7.4    5s
 89882 34961 infeasible   52               - 9790.49993      -   7.5   10s
 137127 51937 15021.3346   36   18          - 10241.9032      -   7.6   15s
 194323 72583 12363.7848   45   20          - 10525.6954      -   7.6   20s
 237964 89073 11021.2822   51   21          - 10697.7421      -   7.6   25s
 284863 107210 11436.1599   36   23          - 10817.8288      -   7.5   30s
 327509 124190 infeasible   49               - 10909.0701      -   7.5   35s
 360916 136502 infeasible   52               - 10976.7928      -   7.5   40s
 393440 148344 12878.3348   36   21          - 11039.5800      -   7.5   45s
 430150 162455 12315.5012   46   23          - 11099.4176      -   7.5   50s
 467654 175304 13905.1432   41   22          - 11165.9502      -   7.5   55s
 508032 189914 infeasible   50               - 11224.5765      -   7.5   60s
 546220 202137 11482.0840   43   21          - 11288.1124      -   7.5   65s
 590726 217355 infeasible   52               - 11348.5714      -   7.5   70s
 621544 227685 12825.8401   44   21          - 11391.5712      -   7.5   75s
 651051 237968 infeasible   49               - 11427.3128      -   7.5   80s
 678139 248258 13669.5264   38   19          - 11460.3314      -   7.5   85s
 702526 256123 11738.2474   40   21          - 11490.4760      -   7.5   90s
 728945 264792 infeasible   35               - 11519.6633      -   7.5   95s
 750623 272090 infeasible   50               - 11545.1981      -   7.5  100s
 786143 283588 infeasible   50               - 11584.2251      -   7.5  105s
 828323 296064 infeasible   49               - 11634.5554      -   7.5  110s
 863974 306608 infeasible   48               - 11673.2684      -   7.5  115s
 900167 316800 24926.9714   51   23          - 11714.2175      -   7.5  120s
 938691 328703 17537.5794   40   20          - 11753.4451      -   7.5  125s
 979417 341112 25644.4150   52   17          - 11794.1565      -   7.5  130s
 1025402 354624 infeasible   50               - 11836.0260      -   7.5  135s
 1068357 366461 infeasible   46               - 11878.1217      -   7.5  140s
 1111625 378502 11940.2937   43   20          - 11921.9336      -   7.5  145s
```

```
 1154718 390597 19437.4415   50   25          - 11962.0713      -   7.5  150s
 1197808 401455 12295.4429   40   21          - 12003.8213      -   7.5  155s
 1238484 412908 infeasible   47               - 12041.3657      -   7.5  160s
 1276257 423544 31521.5837   64   23          - 12074.3179      -   7.5  165s
 1315020 434176 infeasible   49               - 12106.4520      -   7.5  170s
 1354766 445550 12128.4140  121    5          - 12126.6249      -   7.4  175s
 1391950 451267 infeasible  131               - 12126.6490      -   7.4  180s
 1431802 457241 12126.6758  124    3          - 12126.6745      -   7.3  185s
 1469098 460360 12126.7083  127    2          - 12126.7053      -   7.2  190s
 1505534 463343 infeasible  174               - 12126.7085      -   7.3  195s
 1537458 463981 infeasible  180               - 12126.7085      -   7.3  200s
 1573114 463679 infeasible  175               - 12126.7086      -   7.4  205s
 1609596 463784 12126.7086  179    6          - 12126.7086      -   7.5  210s
 1646176 463848 infeasible  181               - 12126.7086      -   7.5  215s
 1683307 463806 12126.7087  174    5          - 12126.7087      -   7.6  220s
 1717175 463373 infeasible  178               - 12126.7087      -   7.7  225s
 1750158 464401 infeasible  180               - 12126.7087      -   7.7  230s
 1787039 464793 infeasible  181               - 12126.7088      -   7.8  235s
 1819044 464441 infeasible  181               - 12126.7088      -   7.8  240s
 1849041 464272 infeasible  178               - 12126.7088      -   7.9  245s
 1881407 463773 infeasible  183               - 12126.7089      -   8.0  250s
 1916565 463771 infeasible  178               - 12126.7089      -   8.0  255s
 1952157 463670 infeasible  178               - 12126.7090      -   8.1  260s
 1987176 463406 infeasible  178               - 12126.7090      -   8.2  265s
 2022705 464325 infeasible  176               - 12126.7090      -   8.2  270s
 2058123 464278 infeasible  178               - 12126.7091      -   8.2  275s
 2091483 463719 infeasible  184               - 12126.7091      -   8.3  280s
 2116440 463678 12126.7091  175    5          - 12126.7091      -   8.4  285s
 2156763 463305 12126.7092  177    5          - 12126.7092      -   8.4  290s
 2190218 463894 infeasible  178               - 12126.7092      -   8.5  295s
 2225053 463473 infeasible  178               - 12126.7092      -   8.5  300s
 2252764 463500 infeasible  178               - 12126.7093      -   8.6  305s
 2292887 463555 infeasible  179               - 12126.7093      -   8.7  310s
 2324328 463111 infeasible  181               - 12126.7093      -   8.8  315s
 2358380 464534 12126.7094  173    6          - 12126.7094      -   8.8  321s
 2389330 465042 12126.7094  171    4          - 12126.7094      -   8.8  325s
 2433576 465135 12126.7095  180    6          - 12126.7094      -   8.8  330s
 2468404 464723 infeasible  182               - 12126.7095      -   8.8  335s
 2498812 464302 12126.7095  180    6          - 12126.7095      -   8.9  340s
 2524631 465407 12126.7096  169    5          - 12126.7095      -   8.9  345s
 2573345 465664 infeasible  175               - 12126.7096      -   8.9  350s
 2624065 464889 infeasible  178               - 12126.7097      -   8.8  355s
 2677211 464263 infeasible  184               - 12126.7097      -   8.6  360s
 2734210 465195 12126.7098  182    6          - 12126.7098      -   8.5  365s
 2786867 464392 12126.7099  175    5          - 12126.7099      -   8.3  370s
 2828694 464569 infeasible  179               - 12126.7099      -   8.2  375s
 2883591 464148 12126.7100  184    6          - 12126.7100      -   8.0  380s
 2949736 463561 infeasible  181               - 12126.7101      -   7.9  385s
 3006990 463598 infeasible  184               - 12126.7101      -   7.7  390s
 3069444 464464 12126.7103  182    6          - 12126.7102      -   7.6  395s
 3125630 463829 12126.7102  182    6          - 12126.7102      -   7.5  400s
 3181221 463266 12126.7104  177    5          - 12126.7103      -   7.3  405s
 3242975 463639 infeasible  178               - 12126.7104      -   7.2  410s
 3296193 463616 infeasible  179               - 12126.7104      -   7.1  415s
 3353223 462965 12126.7105  180    5          - 12126.7105      -   7.0  420s
 3407723 464690 infeasible  176               - 12126.7106      -   6.9  425s
 3462272 464252 infeasible  176               - 12126.7106      -   6.8  430s
 3516936 465189 12126.7107  182    6          - 12126.7106      -   6.7  435s
 3567200 464243 infeasible  178               - 12126.7107      -   6.6  440s
 3625693 464281 infeasible  178               - 12126.7108      -   6.5  445s
 3671895 463822 12126.7109  177    5          - 12126.7108      -   6.4  450s
 3714048 463472 12126.7109  183    6          - 12126.7109      -   6.3  455s
 3775813 463821 infeasible  184               - 12126.7110      -   6.2  460s
 3826985 463175 infeasible  178               - 12126.7110      -   6.2  465s
 3874472 463049 12126.7111  175    5          - 12126.7111      -   6.1  470s
 3926191 463716 12126.7112  175    5          - 12126.7112      -   6.0  475s
 3980763 463806 12126.7112  177    5          - 12126.7112      -   5.9  480s
 4033748 463729 12126.7113  179    5          - 12126.7113      -   5.9  485s
 4085627 463246 12126.7114  178    5          - 12126.7113      -   5.8  490s
 4141679 463459 12126.7115  185    6          - 12126.7114      -   5.7  495s
 4191804 462643 infeasible  181               - 12126.7115      -   5.7  500s
 4250507 463851 12126.7116  179    5          - 12126.7116      -   5.6  505s
 4311489 463069 12126.7117  180    5          - 12126.7117      -   5.5  510s
 4365327 463247 12126.7118  185    6          - 12126.7118      -   5.5  515s
 4417571 463126 12126.7119  180    5          - 12126.7119      -   5.4  520s
```

```
4470491 462729 infeasible  183              - 12126.7119      -   5.3  525s
4531430 464992 12126.7120  177    6         - 12126.7120      -   5.3  530s
4590459 465467 12126.7121  179    6         - 12126.7121      -   5.2  535s
4642861 464586 infeasible  177              - 12126.7121      -   5.2  540s
4705944 465757 12126.7123  170    4         - 12126.7122      -   5.1  545s
4761148 465874 infeasible  178              - 12126.7123      -   5.0  550s
4803281 465106 12126.7123  173    5         - 12126.7123      -   5.0  555s
4852733 464554 infeasible  178              - 12126.7124      -   5.0  560s
4909961 465699 infeasible  182              - 12126.7124      -   4.9  565s
4964201 464830 12126.7125  183    6         - 12126.7125      -   4.9  570s
5012066 464855 12126.7125  176    5         - 12126.7125      -   4.8  575s
5065564 464517 infeasible  180              - 12126.7126      -   4.8  580s
5118210 464145 12126.7127  180    5         - 12126.7126      -   4.7  585s
5168835 466057 12126.7127  174    4         - 12126.7127      -   4.7  590s
5224824 466195 12126.7128  174    5         - 12126.7128      -   4.6  595s
5280371 466859 12126.7130  176    6         - 12126.7128      -   4.6  600s
5335016 466592 12126.7129  177    5         - 12126.7129      -   4.5  605s
5396166 465941 infeasible  177              - 12126.7129      -   4.5  610s
5452607 466405 12126.7130  176    5         - 12126.7130      -   4.5  615s
5504800 465965 12126.7131  177    5         - 12126.7130      -   4.4  620s
5551462 465656 12126.7131  179    5         - 12126.7131      -   4.4  625s
5619933 464851 12126.7132  182    6         - 12126.7132      -   4.3  630s
5673308 465097 12126.7133  175    5         - 12126.7133      -   4.3  635s
5732847 465216 12126.7133  183    6         - 12126.7133      -   4.3  640s
5784149 464505 infeasible  178              - 12126.7134      -   4.2  645s
5841728 464485 12126.7135  179    5         - 12126.7134      -   4.2  650s
5896496 464010 infeasible  187              - 12126.7135      -   4.2  655s
5950241 465505 12126.7136  176    5         - 12126.7136      -   4.1  660s
6002816 465791 infeasible  184              - 12126.7136      -   4.1  665s
6061287 465296 infeasible  184              - 12126.7137      -   4.1  670s
6114826 464705 12126.7138  186    6         - 12126.7138      -   4.0  675s
6167954 464491 12126.7139  179    6         - 12126.7139      -   4.0  680s
6225949 464441 12126.7140  179    5         - 12126.7139      -   4.0  685s
6280461 464936 infeasible  184              - 12126.7140      -   3.9  690s
6337998 464745 12126.7141  180    5         - 12126.7141      -   3.9  695s
6386571 464349 12126.7142  183    6         - 12126.7142      -   3.9  700s
6440247 464655 12126.7144  179    5         - 12126.7143      -   3.8  705s
6489263 463948 infeasible  181              - 12126.7144      -   3.8  710s
6537088 464035 12126.7145  177    5         - 12126.7145      -   3.8  715s
6595023 463869 12126.7146  183    5         - 12126.7146      -   3.8  720s
6649634 463667 infeasible  175              - 12126.7147      -   3.7  725s
6705933 463641 infeasible  175              - 12126.7148      -   3.7  730s
6763583 464340 infeasible  178              - 12126.7148      -   3.7  735s
6819089 463682 12126.7149  177    4         - 12126.7149      -   3.7  740s
6867404 463342 12126.7150  170    5         - 12126.7149      -   3.6  745s
6922637 463598 infeasible  184              - 12126.7150      -   3.6  750s
6983856 463715 12126.7151  182    6         - 12126.7150      -   3.6  755s
7037993 462740 infeasible  187              - 12126.7151      -   3.6  760s
7093291 465075 12126.7152  175    5         - 12126.7152      -   3.5  765s
7147177 464317 12126.7152  181    6         - 12126.7152      -   3.5  770s
7202370 464600 12126.7153  183    6         - 12126.7153      -   3.5  775s
7259690 464346 12126.7154  177    5         - 12126.7154      -   3.5  780s
7312905 463742 12126.7155  183    6         - 12126.7155      -   3.4  785s
7372602 463243 infeasible  178              - 12126.7156      -   3.4  790s
7430259 463671 12126.7156  177    5         - 12126.7156      -   3.4  795s
7484377 463875 infeasible  178              - 12126.7157      -   3.4  800s
7535905 462991 12126.7158  182    6         - 12126.7157      -   3.3  805s
7602813 464072 infeasible  178              - 12126.7158      -   3.3  810s
7659802 464186 infeasible  178              - 12126.7159      -   3.3  815s
7715804 463625 12126.7159  176    5         - 12126.7159      -   3.3  820s
7757183 463673 12126.7160  178    5         - 12126.7160      -   3.3  825s
7825403 462973 12126.7161  183    6         - 12126.7161      -   3.2  830s
7885247 464037 infeasible  179              - 12126.7162      -   3.2  835s
7940810 463514 infeasible  187              - 12126.7162      -   3.2  840s
7992812 463094 12126.7164  178    4         - 12126.7164      -   3.2  845s
8048755 462886 12126.7165  179    5         - 12126.7164      -   3.2  850s
8109466 462887 12126.7165  182    5         - 12126.7165      -   3.1  855s
8163702 463183 12126.7181  173    5         - 12126.7166      -   3.1  860s
8221933 464814 12126.7167  182    5         - 12126.7167      -   3.1  865s
8276251 465023 12126.7167  175    5         - 12126.7167      -   3.1  870s
8336837 465291 12126.7168  174    5         - 12126.7168      -   3.1  875s
8391317 464339 12126.7169  178    5         - 12126.7168      -   3.0  880s
8444447 464483 12126.7169  184    6         - 12126.7169      -   3.0  885s
8502409 464037 12126.7170  180    5         - 12126.7169      -   3.0  890s
8555036 466034 12126.7170  180    6         - 12126.7170      -   3.0  895s
```

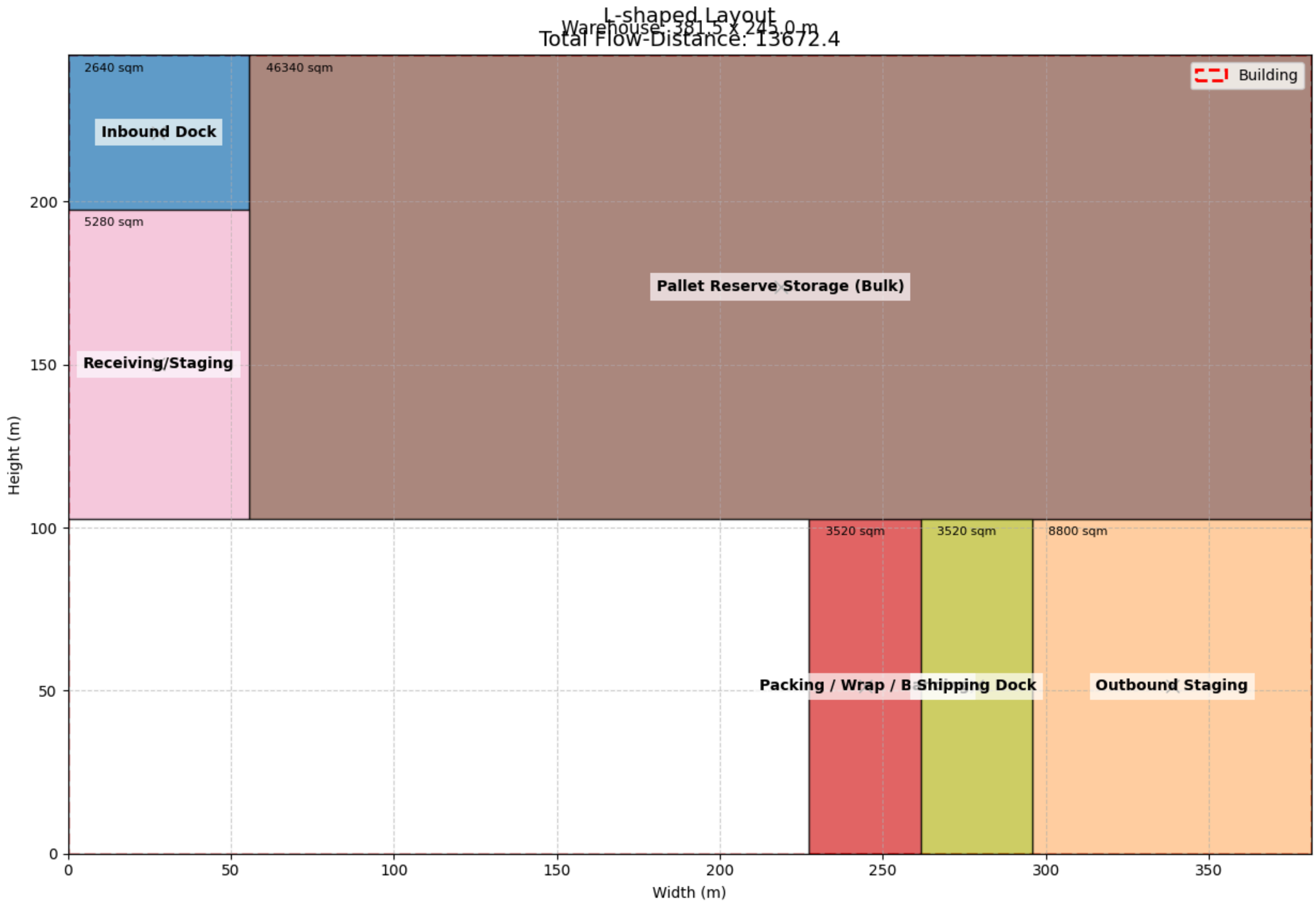
```
8604916 465708 12126.7171  177    5          - 12126.7171      -   3.0  900s
8657202 466018 12126.7171  176    5          - 12126.7171      -   3.0  905s
8708791 466014 infeasible  179               - 12126.7172      -   3.0  910s
8762697 465034 infeasible  181               - 12126.7172      -   2.9  915s
8811864 464374 infeasible  178               - 12126.7173      -   2.9  920s
8854825 464543 infeasible  184               - 12126.7173      -   2.9  925s
8908397 464202 infeasible  186               - 12126.7174      -   2.9  930s
8964438 465243 12126.7177  181    6          - 12126.7175      -   2.9  935s
9022383 465018 12126.7176  177    5          - 12126.7176      -   2.9  940s
9074752 464383 infeasible  179               - 12126.7177      -   2.8  945s
9130800 464130 12126.7178  178    5          - 12126.7177      -   2.8  950s
9188478 464647 12126.7178  179    5          - 12126.7178      -   2.8  955s
9241298 463910 infeasible  187               - 12126.7179      -   2.8  960s
9302953 463644 infeasible  185               - 12126.7180      -   2.8  965s
9356933 463538 12126.7181  177    5          - 12126.7181      -   2.8  970s
9407881 463818 infeasible  178               - 12126.7182      -   2.8  975s
9470292 463082 12126.7183  176    5          - 12126.7182      -   2.7  980s
9527599 463079 infeasible  181               - 12126.7183      -   2.7  985s
9580824 464145 infeasible  178               - 12126.7184      -   2.7  990s
9637537 464041 infeasible  178               - 12126.7184      -   2.7  995s
9697583 463857 12126.7185  184    6          - 12126.7185      -   2.7 1000s
9755222 463301 infeasible  179               - 12126.7186      -   2.7 1005s
9807280 462887 12126.7187  180    5          - 12126.7187      -   2.7 1010s
9856593 463794 12126.7188  178    5          - 12126.7188      -   2.7 1015s
9911047 463386 infeasible  181               - 12126.7188      -   2.6 1020s
9960888 462764 12126.7189  178    5          - 12126.7189      -   2.6 1025s
10014095 462960 12126.7190  186    6          - 12126.7190      -   2.6 1030s
10070784 463113 12126.7191  180    5          - 12126.7191      -   2.6 1035s
10119106 462748 12126.7191  187    6          - 12126.7191      -   2.6 1040s
10172998 463501 12126.7192  175    5          - 12126.7192      -   2.6 1045s
10236813 463955 12126.7193  184    6          - 12126.7193      -   2.6 1050s
10295310 463539 12126.7194  179    5          - 12126.7193      -   2.6 1055s
10352020 464189 12126.7194  177    5          - 12126.7194      -   2.6 1060s
10410420 464117 12126.7195  179    5          - 12126.7195      -   2.5 1065s
10462817 463591 infeasible  180               - 12126.7196      -   2.5 1070s
10521132 463111 infeasible  187               - 12126.7198      -   2.5 1075s
10583950 462879 infeasible  181               - 12126.7201      -   2.5 1080s
10638376 462809 infeasible  180               - 12126.7201      -   2.5 1085s
10692617 463458 infeasible  184               - 12126.7202      -   2.5 1090s
10752378 463099 12126.7203  186    6          - 12126.7203      -   2.5 1095s
10808824 462785 infeasible  180               - 12126.7204      -   2.5 1100s
10863122 462749 infeasible  182               - 12126.7205      -   2.4 1105s
10917419 462445 infeasible  190               - 12126.7206      -   2.4 1110s
10978549 463651 12126.7207  176    5          - 12126.7207      -   2.4 1115s
11035838 463393 12126.7208  177    4          - 12126.7208      -   2.4 1120s
11093851 463502 infeasible  180               - 12126.7209      -   2.4 1125s
11152700 463471 infeasible  181               - 12126.7210      -   2.4 1130s
11207289 463156 12126.7211  183    5          - 12126.7211      -   2.4 1135s
11261432 462476 infeasible  187               - 12126.7212      -   2.4 1140s
11313120 462605 12126.7213  185    6          - 12126.7213      -   2.4 1145s
11377168 462556 12126.7214  183    6          - 12126.7214      -   2.4 1150s
11418648 462354 12126.7214  187    6          - 12126.7214      -   2.4 1155s
11468221 462994 12126.7215  179    5          - 12126.7215      -   2.3 1160s
11526355 462882 infeasible  183               - 12126.7216      -   2.3 1165s
11580402 462733 12126.7217  186    6          - 12126.7217      -   2.3 1170s
11634403 462700 12126.7218  187    6          - 12126.7218      -   2.3 1175s
11690634 462279 12126.7219  183    5          - 12126.7219      -   2.3 1180s
11750343 463024 infeasible  183               - 12126.7220      -   2.3 1185s
11806506 462739 12126.7221  182    4          - 12126.7220      -   2.3 1190s
11860095 462138 12126.7222  184    5          - 12126.7222      -   2.3 1195s
11916576 462462 infeasible  191               - 12126.7222      -   2.3 1200s
11972706 462322 infeasible  185               - 12126.7224      -   2.3 1205s
12028319 462230 12126.7224  185    5          - 12126.7224      -   2.3 1210s
12082981 461893 12126.7225  188    5          - 12126.7225      -   2.2 1215s
12140058 464440 12126.7226  173    5          - 12126.7226      -   2.2 1220s
12192901 464174 12126.7227  180    6          - 12126.7226      -   2.2 1225s
12243757 463675 12126.7228  166    5          - 12126.7227      -   2.2 1230s
12293980 464905 12126.7227  179    6          - 12126.7227      -   2.2 1235s
12333091 464929 infeasible  178               - 12126.7228      -   2.2 1240s
12372340 464235 12126.7228  183    6          - 12126.7228      -   2.2 1245s
12409731 463988 infeasible  184               - 12126.7229      -   2.2 1250s
12446381 463737 12126.7229  175    5          - 12126.7229      -   2.2 1255s
12478346 464451 infeasible  178               - 12126.7229      -   2.2 1260s
12511997 463903 12126.7230  173    5          - 12126.7230      -   2.2 1265s
12549922 463606 infeasible  179               - 12126.7230      -   2.2 1270s
```

12593165	463498	12126.7231	177	5	-	12126.7231	-	2.2	1275s
12624158	463145	infeasible	187		-	12126.7232	-	2.2	1280s
12659649	464772	12126.7238	181	6	-	12126.7232	-	2.2	1285s
12701339	464860	infeasible	177		-	12126.7233	-	2.2	1290s
12742107	464672	infeasible	176		-	12126.7233	-	2.2	1295s
12788269	465417	12126.7234	182	6	-	12126.7234	-	2.1	1300s
12827853	464943	infeasible	178		-	12126.7234	-	2.1	1305s
12872672	464614	12126.7235	176	5	-	12126.7235	-	2.1	1310s
12919762	464799	infeasible	178		-	12126.7235	-	2.1	1315s
12963046	465006	infeasible	179		-	12126.7236	-	2.1	1320s
13010737	464155	infeasible	187		-	12126.7237	-	2.1	1325s
13047422	463549	12126.7238	177	5	-	12126.7238	-	2.1	1330s
13089657	464104	infeasible	181		-	12126.7238	-	2.1	1335s
13131546	463992	12126.7238	177	5	-	12126.7238	-	2.1	1340s
13165880	463374	12126.7239	177	5	-	12126.7239	-	2.1	1345s
13221382	463436	infeasible	179		-	12126.7240	-	2.1	1350s
13263393	463184	infeasible	181		-	12126.7240	-	2.1	1355s
13310141	464431	infeasible	178		-	12126.7241	-	2.1	1360s
13350346	464435	12126.7242	176	4	-	12126.7241	-	2.1	1365s
13393431	464435	infeasible	177		-	12126.7242	-	2.1	1370s
13425300	464284	infeasible	179		-	12126.7242	-	2.1	1375s
13467488	464062	12126.7243	184	6	-	12126.7243	-	2.1	1380s
13500731	463554	12126.7244	182	6	-	12126.7244	-	2.1	1385s
13539950	463537	infeasible	178		-	12126.7244	-	2.1	1390s
13577387	463310	infeasible	180		-	12126.7245	-	2.0	1395s
13619353	463692	12126.7246	176	4	-	12126.7246	-	2.0	1400s
13674053	464217	12126.7246	177	5	-	12126.7246	-	2.0	1405s
13717203	463334	infeasible	179		-	12126.7247	-	2.0	1410s
13768262	463425	infeasible	179		-	12126.7248	-	2.0	1415s
13814941	463533	infeasible	181		-	12126.7249	-	2.0	1420s
13855982	463155	12126.7250	180	5	-	12126.7250	-	2.0	1425s
13898013	463028	infeasible	181		-	12126.7251	-	2.0	1430s
13940260	462839	infeasible	190		-	12126.7251	-	2.0	1435s
13992382	465794	12126.7272	181	6	-	12126.7252	-	2.0	1440s
14037991	465820	infeasible	178		-	12126.7252	-	2.0	1445s
14079455	465561	12126.7253	177	5	-	12126.7253	-	2.0	1450s
14129940	466527	12126.7253	177	6	-	12126.7253	-	2.0	1455s
14170381	466235	infeasible	178		-	12126.7254	-	2.0	1460s
14225734	465876	infeasible	178		-	12126.7254	-	2.0	1465s
14275231	465942	12126.7255	182	6	-	12126.7255	-	2.0	1470s
14324903	465534	12126.7255	177	5	-	12126.7255	-	2.0	1475s
14385268	465552	infeasible	179		-	12126.7256	-	2.0	1480s
14430270	466121	12126.7257	173	5	-	12126.7257	-	2.0	1485s
14479864	466984	12126.7257	177	5	-	12126.7257	-	1.9	1490s
14527459	466684	infeasible	176		-	12126.7258	-	1.9	1495s
14582812	467595	12126.7258	183	6	-	12126.7258	-	1.9	1500s
14629659	466655	12126.7259	174	5	-	12126.7259	-	1.9	1505s
14680825	466803	infeasible	178		-	12126.7259	-	1.9	1510s
H14688073	466825				191006.28238	12126.7259	93.7%	1.9	1511s
H14688168	466786				82742.340352	12126.7259	85.3%	1.9	1511s
H14688496	458093				29203.073166	12126.7259	58.5%	1.9	1511s
14704717	451591	infeasible	133		29203.0732	12126.7332	58.5%	1.9	1515s
14722300	443432	infeasible	140		29203.0732	12127.5415	58.5%	1.9	1520s
14738708	438188	infeasible	61		29203.0732	12130.6481	58.5%	1.9	1525s
14752486	435229	21235.8799	39	19	29203.0732	12136.4819	58.4%	1.9	1530s
14761465	436440	16763.8171	37	24	29203.0732	12145.9389	58.4%	1.9	1537s
14773167	436910	infeasible	54		29203.0732	12156.3714	58.4%	1.9	1540s
H14782013	392340				23398.548072	12164.8435	48.0%	1.9	1544s
14782128	392340	infeasible	50		23398.5481	12164.8670	48.0%	1.9	1545s
14784303	392461	15949.5701	55	20	23398.5481	12167.1228	48.0%	1.9	1550s
14792921	392971	14641.4133	53	18	23398.5481	12176.1997	48.0%	1.9	1555s
14804065	393069	13220.8326	46	25	23398.5481	12188.0222	47.9%	2.0	1560s
14814339	393524	16966.2222	51	20	23398.5481	12199.2829	47.9%	2.0	1565s
14825399	394386	18049.5470	54	16	23398.5481	12209.3309	47.8%	2.0	1570s
14843347	396173	infeasible	50		23398.5481	12222.7225	47.8%	2.0	1575s
14863936	397287	cutoff	52		23398.5481	12243.8105	47.7%	2.0	1580s
14890483	398215	13322.3349	56	17	23398.5481	12272.9385	47.5%	2.0	1585s
14916649	399281	16186.1521	46	21	23398.5481	12300.1037	47.4%	2.0	1590s
14939980	400647	13059.5329	42	16	23398.5481	12321.3883	47.3%	2.0	1595s
14961138	401793	infeasible	52		23398.5481	12339.2417	47.3%	2.0	1600s
14983140	402487	12431.7748	57	17	23398.5481	12359.7400	47.2%	2.0	1605s
15005561	403119	infeasible	53		23398.5481	12380.7506	47.1%	2.0	1610s
15025858	403922	infeasible	43		23398.5481	12400.6851	47.0%	2.0	1615s
15044213	404321	14742.1430	50	18	23398.5481	12416.7692	46.9%	2.1	1620s
15059617	404803	infeasible	51		23398.5481	12429.5034	46.9%	2.1	1625s

```
 15074657 404984 infeasible   50      23398.5481 12442.9561  46.8%   2.1 1630s
 15085576 405458 infeasible   43      23398.5481 12453.6253  46.8%   2.1 1635s
 15104905 406205 13051.2608   35   20 23398.5481 12471.1990  46.7%   2.1 1640s
 15120442 406834 14753.1145   50   21 23398.5481 12484.5436  46.6%   2.1 1645s
 15134995 407371 infeasible   46      23398.5481 12496.9402  46.6%   2.1 1650s
 15152303 408325 14278.0671   45   18 23398.5481 12511.0870  46.5%   2.1 1655s
 15173307 408971 18906.7351   43   21 23398.5481 12529.3038  46.5%   2.1 1660s
 15192766 409413 infeasible   51      23398.5481 12546.8202  46.4%   2.1 1665s
 15211984 409805 infeasible   44      23398.5481 12562.3337  46.3%   2.1 1670s
 15230286 410203 infeasible   48      23398.5481 12578.6174  46.2%   2.1 1675s
 15248438 410721 22258.8826   55   18 23398.5481 12595.6102  46.2%   2.1 1680s
 15267169 411419 infeasible   60      23398.5481 12609.3595  46.1%   2.2 1685s
 15283541 411661 18512.6529   49   19 23398.5481 12623.8527  46.0%   2.2 1690s
 15301054 412096 12969.6416   44   21 23398.5481 12639.6314  46.0%   2.2 1695s
 15319226 412600 15039.6819   45   23 23398.5481 12654.8905  45.9%   2.2 1700s
 15334884 412950 infeasible   62      23398.5481 12666.5907  45.9%   2.2 1705s
 15354205 413275 18381.6891   52   22 23398.5481 12681.3311  45.8%   2.2 1710s
 15370167 413626 12718.1118   69    9 23398.5481 12694.0458  45.7%   2.2 1715s
 15387693 413913 infeasible   45      23398.5481 12708.5224  45.7%   2.2 1720s
 15404119 414510 infeasible   43      23398.5481 12722.6931  45.6%   2.2 1725s
 15415531 415024 infeasible   41      23398.5481 12731.4377  45.6%   2.2 1730s
 15430248 415518     cutoff   43      23398.5481 12743.1000  45.5%   2.2 1735s
 15447365 415766 infeasible   53      23398.5481 12755.6672  45.5%   2.2 1740s
 15462707 416127 infeasible   42      23398.5481 12768.0671  45.4%   2.2 1745s
 15480179 416568 infeasible   41      23398.5481 12780.8853  45.4%   2.2 1750s
 15490976 416868 12805.2537   45   19 23398.5481 12788.6017  45.3%   2.3 1755s
 15505916 417315 infeasible   50      23398.5481 12800.4460  45.3%   2.3 1760s
H15520261 417575                    23398.548058 12811.6331  45.2%   2.3 1763s
 15523620 417720 13700.7199   50   19 23398.5481 12814.0763  45.2%   2.3 1765s
 15540732 418035 infeasible   42      23398.5481 12827.7486  45.2%   2.3 1770s
 15556639 418672 13860.8243   46   19 23398.5481 12839.5849  45.1%   2.3 1775s
 15572593 418752 13927.4171   52   16 23398.5481 12852.2625  45.1%   2.3 1780s
 15589161 419304 infeasible   40      23398.5481 12865.2982  45.0%   2.3 1785s
 15603583 419446 infeasible   44      23398.5481 12878.5050  45.0%   2.3 1790s
 15619331 419984 18684.4383   42   21 23398.5481 12891.3644  44.9%   2.3 1795s
 15635540 420561 infeasible   41      23398.5481 12903.9017  44.9%   2.3 1800s
 15650811 420862 17308.7368   52   21 23398.5481 12917.1299  44.8%   2.3 1805s
 15665747 421267 infeasible   52      23398.5481 12928.8720  44.7%   2.3 1810s
 15681552 421566 18688.5414   49   18 23398.5481 12941.4042  44.7%   2.3 1815s
 15696074 421963 infeasible   54      23398.5481 12953.4476  44.6%   2.3 1820s
H15710109 422104                    23398.548053 12964.7304  44.6%   2.3 1824s
 15710962 422117 15532.2009   49   16 23398.5481 12965.9957  44.6%   2.3 1825s
 15724130 422338 16634.7529   56   21 23398.5481 12976.6591  44.5%   2.3 1830s
 15731892 422452 infeasible   46      23398.5481 12983.2188  44.5%   2.4 1835s
 15747482 422727 infeasible   44      23398.5481 12995.7075  44.5%   2.4 1840s
 15763831 423087 infeasible   48      23398.5481 13008.9382  44.4%   2.4 1845s
 15778844 423177 19032.6960   42   21 23398.5481 13020.6347  44.4%   2.4 1850s
 15787281 423268 14809.8754   47   23 23398.5481 13027.7881  44.3%   2.4 1855s
 15801180 423513 infeasible   40      23398.5481 13039.3096  44.3%   2.4 1860s
 15816170 423734 infeasible   43      23398.5481 13052.5801  44.2%   2.4 1865s
 15830374 424109 13063.5864   33   21 23398.5481 13063.3223  44.2%   2.4 1870s
 15845068 424098 16366.2880   52   21 23398.5481 13075.1357  44.1%   2.4 1875s
 15858651 425029 15761.8265   49   19 23398.5481 13083.3020  44.1%   2.4 1880s
 15871863 425248 infeasible   52      23398.5481 13094.1213  44.0%   2.4 1885s
 15884451 424901 15143.7193   42   19 23398.5481 13105.3026  44.0%   2.4 1890s
 15897340 424877 21104.9522   51   21 23398.5481 13116.0365  43.9%   2.4 1895s
 15910426 425180 21282.2095   43   20 23398.5481 13126.1873  43.9%   2.4 1900s
 15922954 425549 infeasible   51      23398.5481 13136.5756  43.9%   2.4 1905s
 15934774 425849 infeasible   40      23398.5481 13146.1923  43.8%   2.4 1910s
 15945380 426142 15777.2294   53   14 23398.5481 13154.6733  43.8%   2.4 1915s
 15958134 426372 infeasible   52      23398.5481 13164.4891  43.7%   2.4 1920s
 15969149 426683 14129.3615   52   15 23398.5481 13173.7819  43.7%   2.4 1925s
 15980288 426803 14275.8860   48   17 23398.5481 13182.3691  43.7%   2.4 1930s
 15991264 427078 15715.0003   47   19 23398.5481 13191.3737  43.6%   2.5 1935s
 16003142 427345 22573.0573   57   21 23398.5481 13201.0254  43.6%   2.5 1940s
 16015106 427539 infeasible   48      23398.5481 13209.9442  43.5%   2.5 1945s
 16027015 427821 13249.5054   52   16 23398.5481 13219.4217  43.5%   2.5 1950s
 16038437 428114 infeasible   53      23398.5481 13228.9927  43.5%   2.5 1955s
 16049451 428213 infeasible   63      23398.5481 13237.8804  43.4%   2.5 1960s
 16060707 428481 infeasible   51      23398.5481 13246.0392  43.4%   2.5 1965s
 16071580 428587 16495.6027   54   23 23398.5481 13254.9616  43.4%   2.5 1970s
 16082001 428577 infeasible   49      23398.5481 13263.5159  43.3%   2.5 1975s
H16085810 428610                    23398.548047 13266.8569  43.3%   2.5 1977s
H16088807 428615                    23398.548038 13268.5340  43.3%   2.5 1979s
 16089956 428625 14420.7274   45   19 23398.5480 13269.6665  43.3%   2.5 1981s
```

```
 16097926 428822 22835.3355   54   23 23398.5480 13276.8564  43.3%   2.5 1985s
 16108070 429125 infeasible   59      23398.5480 13284.7630  43.2%   2.5 1990s
 16118760 429231 infeasible   60      23398.5480 13293.2014  43.2%   2.5 1995s
 16129277 429552 infeasible   38      23398.5480 13302.0470  43.2%   2.5 2000s
 16141088 429790 17020.5299   49   21 23398.5480 13310.8194  43.1%   2.5 2005s
 16150676 430225 infeasible   44      23398.5480 13318.5017  43.1%   2.5 2010s
 16155459 430475 infeasible   49      23398.5480 13322.0201  43.1%   2.5 2015s
 16165573 430661 infeasible   50      23398.5480 13329.1567  43.0%   2.5 2020s
 16175641 430713 infeasible   51      23398.5480 13338.6254  43.0%   2.5 2025s
 16185699 430946 14239.2790   47   21 23398.5480 13346.2353  43.0%   2.5 2030s
 16195565 431271 infeasible   49      23398.5480 13353.7169  42.9%   2.5 2035s
 16206349 431511 17332.7587   44   15 23398.5480 13362.0315  42.9%   2.5 2040s
 16215607 431465 13881.9708   60   16 23398.5480 13370.7970  42.9%   2.5 2045s
 16226284 431599 15126.2789   61   18 23398.5480 13379.8758  42.8%   2.5 2050s
 16235227 431769 infeasible   98      23398.5480 13387.0310  42.8%   2.5 2055s
*16242000 234674             131    17251.461727 13391.0217  22.4%   2.5 2059s
 16242886 234606     cutoff   46      17251.4617 13391.7831  22.4%   2.5 2060s
 16249415 234567     cutoff   56      17251.4617 13398.0397  22.3%   2.5 2065s
 16256694 234087 infeasible   60      17251.4617 13406.7054  22.3%   2.6 2070s
 16264688 233732 infeasible   42      17251.4617 13415.3949  22.2%   2.6 2075s
 16273349 233298 15153.2659   56   21 17251.4617 13423.8050  22.2%   2.6 2080s
 16281598 232940 15079.7911   60   17 17251.4617 13433.6452  22.1%   2.6 2085s
 16289011 232487 infeasible   50      17251.4617 13440.8003  22.1%   2.6 2090s
 16296859 231865 13451.4826   35   17 17251.4617 13449.8471  22.0%   2.6 2095s
 16304491 231613     cutoff   40      17251.4617 13458.1573  22.0%   2.6 2100s
 16312384 231201 infeasible   49      17251.4617 13466.1296  21.9%   2.6 2105s
 16320360 230849 13939.3890   58   19 17251.4617 13474.6757  21.9%   2.6 2110s
 16328202 230366 13801.1516   41   17 17251.4617 13483.2450  21.8%   2.6 2115s
 16336038 229932 14372.9454   56   15 17251.4617 13492.3188  21.8%   2.6 2120s
 16343764 229667 infeasible   54      17251.4617 13500.8620  21.7%   2.6 2125s
 16351449 229223 13838.8664   62   18 17251.4617 13509.1258  21.7%   2.6 2130s
 16359278 228819 infeasible   56      17251.4617 13517.2249  21.6%   2.6 2135s
*16364241 24695             141    13703.962190 13523.2143  1.32%   2.6 2138s
 16366042 23776     cutoff   49      13703.9622 13529.3369  1.27%   2.6 2140s
 16370805 19547     cutoff   60      13703.9622 13553.1172  1.10%   2.6 2145s
 16376212 15260 infeasible   45      13703.9622 13581.6140  0.89%   2.6 2150s
*16384040  7584             137    13685.217383 13623.3539  0.45%   2.6 2154s
 16384501  6479 infeasible   55      13685.2174 13625.0401  0.44%   2.6 2155s
H16384654  6477                    13685.217364 13625.1491  0.44%   2.6 2155s
H16386464  5436                    13685.217354 13638.0841  0.34%   2.6 2157s
H16386601  5342                    13684.638641 13638.6361  0.34%   2.6 2157s
 16388539  4026     cutoff   39      13684.6386 13650.7752  0.25%   2.6 2160s
H16388652  2080                    13672.378490 13651.5464  0.15%   2.6 2160s

Cutting planes:
  Gomory: 23
  MIR: 10
  Flow cover: 136
  Inf proof: 7
  RLT: 316

Explored 16391212 nodes (42558258 simplex iterations) in 2164.26 seconds (595.26 work units)
Thread count was 12 (of 12 available processors)

Solution count 10: 13672.4 13672.4 13684.6 ... 29203.1

Optimal solution found (tolerance 1.00e-04)
Best objective 1.367237848974e+04, best bound 1.367237540129e+04, gap 0.0000%

--- L-SHAPED LAYOUT ---
Building dimensions: 381.5 x 245.0 m
Total area: 93466.7 sqm
Objective (total flow-distance): 13672.4
```

```
C:\Users\chant\AppData\Local\Temp\ipykernel_34796\398782425.py:167: MatplotlibDeprecationWarning: The get_cma
p function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` o
r ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colors = plt.cm.get_cmap('tab20', len(selected_depts))
```



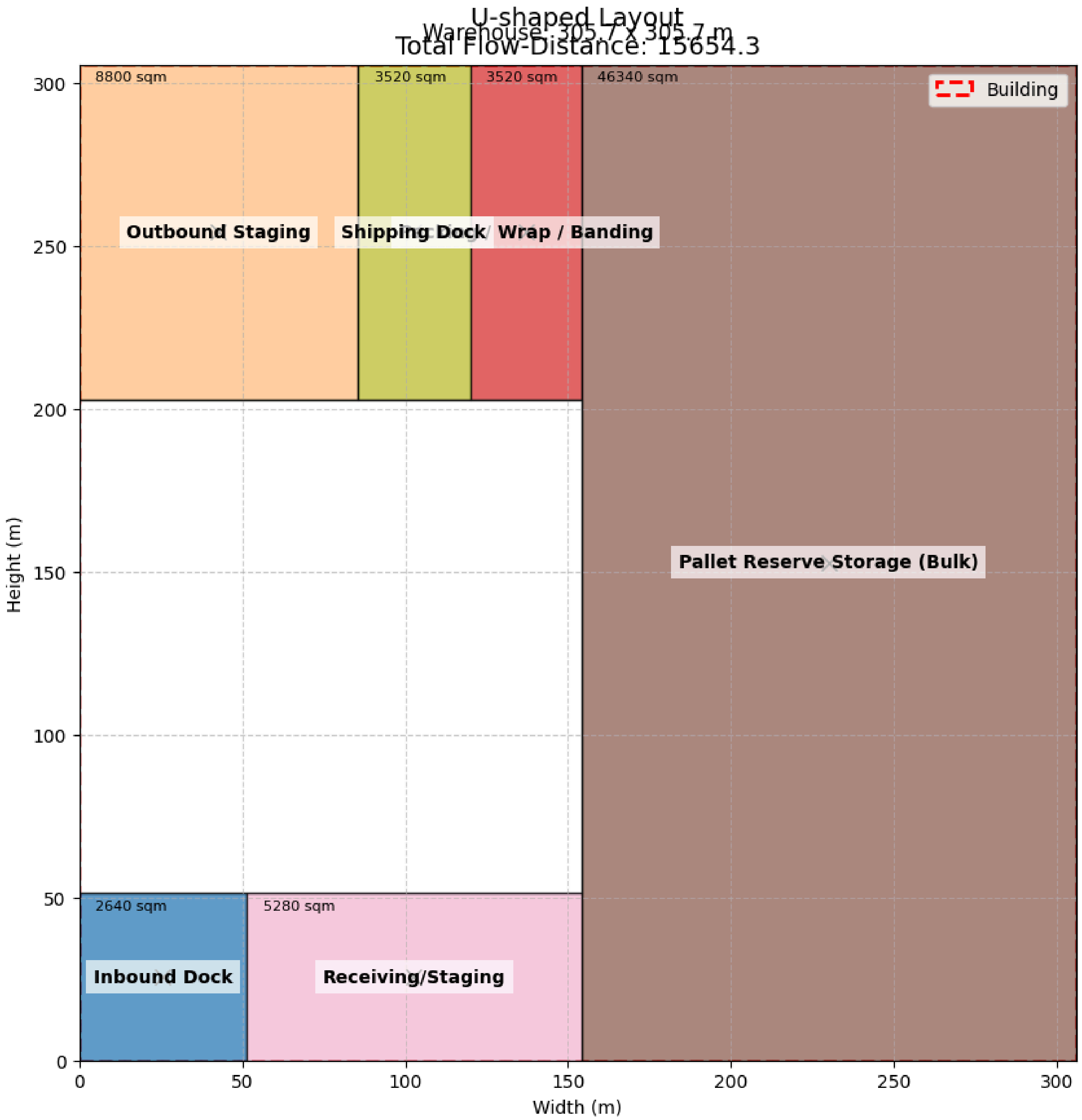

```
In [ ]: # Run for all layouts
        for layout_type in [ "U-shaped"]:
            selected_depts = load_data()
            solve_layout(layout_type, total_area=total_area, selected_depts=selected_depts, departments=departments,
```

```
Set parameter Username
Set parameter LicenseID to value 2718175
Academic license - for non-commercial use only - expires 2026-10-05
Set parameter NonConvex to value 2
Inbound Dock Inbound Dock
Inbound Dock Outbound Staging
Inbound Dock Packing / Wrap / Banding
Inbound Dock Pallet Reserve Storage (Bulk)
Inbound Dock Receiving/Staging
Inbound Dock Shipping Dock
Inbound Dock Phantom
Outbound Staging Inbound Dock
Outbound Staging Outbound Staging
Outbound Staging Packing / Wrap / Banding
Outbound Staging Pallet Reserve Storage (Bulk)
Outbound Staging Receiving/Staging
Outbound Staging Shipping Dock
Outbound Staging Phantom
Packing / Wrap / Banding Inbound Dock
Packing / Wrap / Banding Outbound Staging
Packing / Wrap / Banding Packing / Wrap / Banding
Packing / Wrap / Banding Pallet Reserve Storage (Bulk)
Packing / Wrap / Banding Receiving/Staging
Packing / Wrap / Banding Shipping Dock
Packing / Wrap / Banding Phantom
Pallet Reserve Storage (Bulk) Inbound Dock
Pallet Reserve Storage (Bulk) Outbound Staging
Pallet Reserve Storage (Bulk) Packing / Wrap / Banding
Pallet Reserve Storage (Bulk) Pallet Reserve Storage (Bulk)
Pallet Reserve Storage (Bulk) Receiving/Staging
Pallet Reserve Storage (Bulk) Shipping Dock
Pallet Reserve Storage (Bulk) Phantom
Receiving/Staging Inbound Dock
Receiving/Staging Outbound Staging
Receiving/Staging Packing / Wrap / Banding
Receiving/Staging Pallet Reserve Storage (Bulk)
Receiving/Staging Receiving/Staging
Receiving/Staging Shipping Dock
Receiving/Staging Phantom
Shipping Dock Inbound Dock
Shipping Dock Outbound Staging
Shipping Dock Packing / Wrap / Banding
Shipping Dock Pallet Reserve Storage (Bulk)
Shipping Dock Receiving/Staging
Shipping Dock Shipping Dock
Shipping Dock Phantom
Phantom Inbound Dock
Phantom Outbound Staging
Phantom Packing / Wrap / Banding
Phantom Pallet Reserve Storage (Bulk)
Phantom Receiving/Staging
Phantom Shipping Dock
Phantom Phantom
Total area for U-shaped: 93466.66666666667
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (win64 - Windows 11+.0 (26200.2))

CPU model: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Non-default parameters:
NonConvex  2

Optimize a model with 372 rows, 254 columns and 1036 nonzeros
Model fingerprint: 0xe34ee067
Model has 8 quadratic constraints
Variable types: 156 continuous, 98 integer (98 binary)
Coefficient statistics:
  Matrix range     [3e-01, 1e+04]
  QMatrix range    [1e+00, 1e+00]
  Objective range  [4e+00, 2e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
  QRHS range       [3e+03, 9e+04]
Presolve removed 222 rows and 139 columns
Presolve time: 0.01s
```

```
Presolved: 236 rows, 135 columns, 639 nonzeros
Presolved model has 1 quadratic constraint(s)
Presolved model has 20 bilinear constraint(s)
Warning: Model contains variables with very large bounds participating
         in product terms.
         Presolve was not able to compute smaller bounds for these variables.
         Consider bounding these variables or reformulating the model.


Solving non-convex MIQCP

Variable types: 62 continuous, 73 integer (73 binary)

Root relaxation: objective 2.102737e+01, 78 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   21.02737    0   39          -   21.02737      -     -    0s
     0     0  392.24303    0   38          -  392.24303      -     -    0s
     0     0  392.24303    0   37          -  392.24303      -     -    0s
     0     0  537.26412    0   35          -  537.26412      -     -    0s
     0     0  707.31466    0   39          -  707.31466      -     -    0s
     0     0  707.31466    0   39          -  707.31466      -     -    0s
     0     0  716.03843    0   39          -  716.03843      -     -    0s
     0     0  716.03843    0   39          -  716.03843      -     -    0s
     0     0  716.03843    0   39          -  716.03843      -     -    0s
     0     1  716.03843    0   39          -  716.03843      -     -    0s
 39847 15468 infeasible   53               - 9403.20869      -   6.4    5s
 121578 45085 infeasible   63               - 10843.5379      -   6.3   10s
 197484 72930 23754.2009   59   20          - 11513.5826      -   6.3   15s
 266132 96757 infeasible  103               - 11872.6394      -   6.4   20s
 333333 120704 12896.2422   62   21          - 12118.6904      -   6.3   25s
 380974 137503 19940.7538   66   16          - 12288.5227      -   6.3   30s
 435510 156554 16665.4870   51   18          - 12442.8822      -   6.3   35s
 491044 176488 infeasible   59               - 12580.3583      -   6.3   40s
 543943 193833 13525.8217   56   22          - 12685.4340      -   6.2   45s
 594430 206521 15623.1814   56   14          - 12795.6002      -   6.2   50s
 641889 215967 infeasible   60               - 12887.3730      -   6.2   55s
 686354 225398 21388.1873   58   18          - 12965.0881      -   6.3   60s
 731481 234269 22736.3533   55   21          - 13043.3129      -   6.3   65s
 772052 243777 infeasible   58               - 13107.4735      -   6.4   70s
 810528 252561 13535.3348   60   22          - 13160.4365      -   6.4   75s
 847308 261340 13583.3430   55   20          - 13213.8734      -   6.4   80s
 885794 270249 infeasible   73               - 13266.9405      -   6.5   85s
 926454 280062 13333.8036   57   16          - 13321.2516      -   6.5   90s
 967331 290366 14856.6574   61   17          - 13370.2126      -   6.5   95s
 1005010 298846 18117.4468   57   12          - 13418.9562      -   6.5  100s
 1043415 307698 infeasible   59               - 13468.7023      -   6.6  105s
 1081763 315944 infeasible   53               - 13520.2279      -   6.6  110s
 1122175 324882 16821.6805   60   20          - 13572.8045      -   6.6  115s
 1163232 333852 infeasible   73               - 13624.5196      -   6.6  120s
 1202224 343542 23498.9987   55   18          - 13666.8464      -   6.6  125s
 1234881 350820 17466.2638   55   16          - 13704.6901      -   6.6  130s
 1273301 359726 15906.2402   61   16          - 13750.1163      -   6.7  135s
 1311803 368266 24395.1800   62   24          - 13796.3234      -   6.7  140s
 1348909 376988 infeasible   53               - 13833.9323      -   6.7  145s
 1384566 385041 infeasible   76               - 13874.6144      -   6.7  150s
 1420409 392818 25924.3980   55   16          - 13914.3082      -   6.7  155s
 1456261 401327 16789.9077   50   22          - 13950.3803      -   6.7  160s
 1495554 409734 16702.2966   56   19          - 13988.5606      -   6.7  165s
 1530866 417355 infeasible   56               - 14021.3499      -   6.7  170s
 1563665 423914 21089.1012  141    8          - 14055.3234      -   6.7  175s
 1596926 430274 19225.3409   51   16          - 14089.4568      -   6.7  180s
 1631870 437859 15628.9473   59   18          - 14120.0586      -   6.8  185s
 1664502 444252 16559.2063   47   16          - 14150.6153      -   6.8  190s
 1697803 450829 17805.2750   62   16          - 14179.7002      -   6.8  195s
 1727636 456731 17082.1801   65   23          - 14207.5819      -   6.8  200s
 1759290 463379 20252.9772   57   20          - 14237.3633      -   6.8  205s
 1794797 469404 19980.7822   59   13          - 14274.0127      -   6.8  210s
 1829483 475081 14373.8951   63   12          - 14307.2635      -   6.8  215s
 1862384 480825 14895.1699   55   18          - 14336.5846      -   6.8  220s
 1896577 487226 37089.5385   68   20          - 14369.1907      -   6.8  225s
 1932775 493308 17753.1632   61   22          - 14403.9120      -   6.8  230s
 1964424 498462 15133.5287   73   13          - 14434.6550      -   6.8  235s
```

```
 2000201 503538 25759.2339   60   17           - 14468.2939      -   6.8  240s
 2034898 508732 25623.4822   64   16           - 14502.5677      -   6.9  245s
 2067811 514315 28585.4223   68   20           - 14531.5302      -   6.9  250s
 2101282 518727 infeasible   57                - 14562.4571      -   6.9  255s
 2131948 523395 infeasible   62                - 14590.9547      -   6.9  260s
 2164202 528442 24318.5296   66   20           - 14619.3486      -   6.9  265s
 2196597 532766 14865.6343   56   20           - 14651.4995      -   6.9  270s
 2230691 537372 22890.3982   57   17           - 14682.7439      -   6.9  275s
 2265042 542544 infeasible   62                - 14711.3815      -   6.9  280s
 2299019 546973 infeasible   55                - 14741.1762      -   6.9  285s
 2331912 551684 infeasible   60                - 14769.5494      -   6.9  290s
 2360639 555329 19268.0859   55   16           - 14796.2471      -   6.9  295s
 2392439 559788 18441.4137   58   17           - 14823.7412      -   6.9  300s
 2426223 564588 infeasible   61                - 14855.0674      -   6.9  305s
 2459080 569224 21669.2644   62   18           - 14885.7357      -   6.9  310s
 2491607 573798 17043.3471   61   19           - 14914.4941      -   6.9  315s
 2525660 578114 24995.1275   63   18           - 14945.5111      -   7.0  320s
 2559910 583334 15655.1152   60   13           - 14977.6773      -   7.0  325s
 2593029 587888 34552.5624   65   18           - 15006.1033      -   7.0  330s
 2625596 592402 15173.3492   65   14           - 15037.0941      -   7.0  335s
 2657418 596962 18638.4740   55   20           - 15062.9292      -   7.0  340s
 2689725 601281 infeasible   55                - 15091.6649      -   7.0  345s
 2723477 606469 18208.7433   63   22           - 15118.5662      -   7.0  350s
 2753675 610712 15149.8235   69   20           - 15143.7730      -   7.0  355s
 2784931 614737 infeasible   65                - 15171.5437      -   7.0  360s
 2815181 618033 23680.1450   56   16           - 15198.5542      -   7.0  365s
 2846808 621846 25948.4984   48   20           - 15225.9889      -   7.0  370s
 2880468 625806 16114.1782   60   18           - 15253.6823      -   7.0  375s
 2912021 629445 18667.9140   66   18           - 15280.6542      -   7.0  380s
 2943482 633281 18130.3495   66   21           - 15306.5727      -   7.0  385s
 2975185 637302 infeasible   60                - 15332.4726      -   7.0  390s
 3004913 640524 infeasible   51                - 15356.8349      -   7.0  395s
 3040035 644821 16361.6007   52   16           - 15384.1766      -   7.0  400s
 3073525 647120 19577.9530   52   19           - 15387.1540      -   7.0  405s
 3099732 649858 20763.3124   60   19           - 15407.5616      -   7.0  410s
 3127745 653233 infeasible   74                - 15428.9830      -   7.0  415s
 3159989 657291 19219.8817   68   16           - 15453.4306      -   7.0  420s
 3193083 661082 15768.0268   62   14           - 15477.4707      -   7.0  425s
 3226967 664806 17952.8005   46   17           - 15505.3448      -   7.0  430s
 3258734 668581 infeasible   63                - 15530.6617      -   7.0  435s
 3288831 671962 18541.8198   64   14           - 15554.3913      -   7.0  440s
*3309921 95970             144    16255.187537 15569.2979  4.22%   7.0  444s
 3310718 95111     cutoff   61      16255.1875 15571.3514  4.21%   7.0  446s
H3315279 93764                    16255.187520 15580.7643  4.15%   7.0  446s
H3316947 93743                    16255.187500 15585.3213  4.12%   7.0  447s
 3323982 90792 15743.6580   61   18 16255.1875 15600.9327  4.02%   7.0  450s
 3332668 88402 15872.1402   72   19 16255.1875 15625.2457  3.88%   7.0  455s
*3345209   904             133    15654.322812 15653.1405  0.01%   7.0  459s

Cutting planes:
  Gomory: 12
  Implied bound: 4
  MIR: 3
  Flow cover: 16
  Inf proof: 5
  RLT: 34
  Relax-and-lift: 1
  BQP: 1

Explored 3345251 nodes (23458338 simplex iterations) in 459.83 seconds (139.73 work units)
Thread count was 12 (of 12 available processors)

Solution count 3: 15654.3 15654.3 16255.2

Optimal solution found (tolerance 1.00e-04)
Best objective 1.565432281205e+04, best bound 1.565325947393e+04, gap 0.0068%

--- U-SHAPED LAYOUT ---
Building dimensions: 305.7 x 305.7 m
Total area: 93466.7 sqm
Objective (total flow-distance): 15654.3
```

```
C:\Users\ruben\AppData\Local\Temp\ipykernel_7940\398782425.py:167: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colors = plt.cm.get_cmap('tab20', len(selected_depts))
```



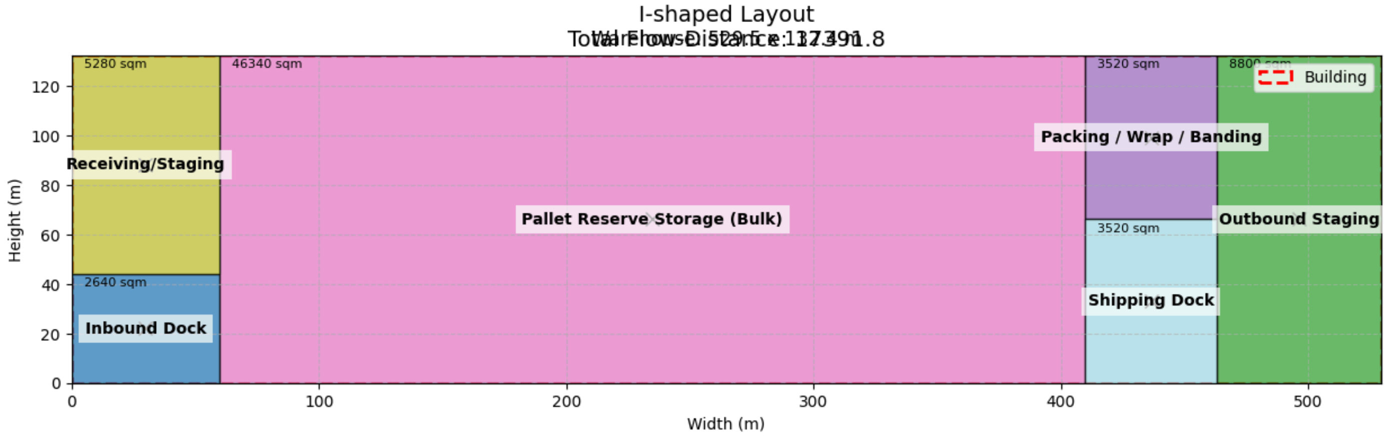
In [ ]:

```python
# Run for all layouts
for layout_type in [ "I-shaped"]:
    selected_depts = load_data()
    solve_layout(layout_type, total_area=total_area, selected_depts=selected_depts, departments=departments,
```

```
Set parameter NonConvex to value 2
Inbound Dock Inbound Dock
Inbound Dock Outbound Staging
Inbound Dock Packing / Wrap / Banding
Inbound Dock Pallet Reserve Storage (Bulk)
Inbound Dock Receiving/Staging
Inbound Dock Shipping Dock
Outbound Staging Inbound Dock
Outbound Staging Outbound Staging
Outbound Staging Packing / Wrap / Banding
Outbound Staging Pallet Reserve Storage (Bulk)
Outbound Staging Receiving/Staging
Outbound Staging Shipping Dock
Packing / Wrap / Banding Inbound Dock
Packing / Wrap / Banding Outbound Staging
Packing / Wrap / Banding Packing / Wrap / Banding
Packing / Wrap / Banding Pallet Reserve Storage (Bulk)
Packing / Wrap / Banding Receiving/Staging
Packing / Wrap / Banding Shipping Dock
Pallet Reserve Storage (Bulk) Inbound Dock
Pallet Reserve Storage (Bulk) Outbound Staging
Pallet Reserve Storage (Bulk) Packing / Wrap / Banding
Pallet Reserve Storage (Bulk) Pallet Reserve Storage (Bulk)
Pallet Reserve Storage (Bulk) Receiving/Staging
Pallet Reserve Storage (Bulk) Shipping Dock
Receiving/Staging Inbound Dock
Receiving/Staging Outbound Staging
Receiving/Staging Packing / Wrap / Banding
Receiving/Staging Pallet Reserve Storage (Bulk)
Receiving/Staging Receiving/Staging
Receiving/Staging Shipping Dock
Shipping Dock Inbound Dock
Shipping Dock Outbound Staging
Shipping Dock Packing / Wrap / Banding
Shipping Dock Pallet Reserve Storage (Bulk)
Shipping Dock Receiving/Staging
Shipping Dock Shipping Dock
Total area for I-shaped: 70100
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (win64 - Windows 11+.0 (26200.2))

CPU model: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Non-default parameters:
NonConvex  2

Optimize a model with 276 rows, 194 columns and 761 nonzeros
Model fingerprint: 0x992e7154
Model has 7 quadratic constraints
Variable types: 122 continuous, 72 integer (72 binary)
Coefficient statistics:
  Matrix range     [5e-01, 1e+04]
  QMatrix range    [1e+00, 1e+00]
  Objective range  [4e+00, 2e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
  QRHS range       [3e+03, 7e+04]
Presolve removed 150 rows and 93 columns
Presolve time: 0.01s
Presolved: 224 rows, 124 columns, 596 nonzeros
Presolved model has 23 bilinear constraint(s)
Warning: Model contains variables with very large bounds participating
         in product terms.
         Presolve was not able to compute smaller bounds for these variables.
         Consider bounding these variables or reformulating the model.


Solving non-convex MIQCP

Variable types: 65 continuous, 59 integer (59 binary)

Root relaxation: objective 1.118464e+02, 64 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  111.84638    0   38          -  111.84638      -     -    0s
     0     0  221.93344    0   50          -  221.93344      -     -    0s
     0     0  282.46422    0   47          -  282.46422      -     -    0s
     0     0  282.46422    0   47          -  282.46422      -     -    0s
     0     0  282.46422    0   43          -  282.46422      -     -    0s
     0     0  282.46422    0   36          -  282.46422      -     -    0s
     0     0  282.46422    0   42          -  282.46422      -     -    0s
     0     0  282.46422    0   46          -  282.46422      -     -    0s
     0     0 2439.50611    0   31          - 2439.50611      -     -    0s
     0     2 2439.50611    0   31          - 2439.50611      -     -    0s
*19468  8042             110    30817.747265 11354.6090  63.2%   6.5    4s
H20431  7933                    30817.678280 11388.5584  63.0%   6.5    4s
*21058  8094              95    29846.274174 11398.2197  61.8%   6.5    4s
H22761  8587                    29840.087956 11471.1831  61.6%   6.5    4s
 27972 10389 25412.3817   68   17 29840.0880 11637.6183  61.0%   6.9    5s
H35391  7289                    17819.522559 11926.7202  33.1%   7.4    6s
 40921  7111 infeasible   96      17819.5226 12005.4342  32.6%   7.2   10s
 70275  1533 infeasible   96      17819.5226 14793.3726  17.0%   7.7   15s
*79152   662             152    17552.810233 16808.2488  4.24%   7.6   16s
*79601   597             194    17449.160080 16964.9586  2.77%   7.6   16s
H80972   475                    17391.837769 17118.0104  1.57%   7.5   17s

Cutting planes:
  Gomory: 13
  Implied bound: 1
  MIR: 7
  Flow cover: 30
  Inf proof: 1
  RLT: 126

Explored 82215 nodes (615082 simplex iterations) in 17.22 seconds (4.01 work units)
Thread count was 12 (of 12 available processors)

Solution count 8: 17391.8 17449.2 17552.8 ... 30817.7

Optimal solution found (tolerance 1.00e-04)
Best objective 1.739183776903e+04, best bound 1.739183776903e+04, gap 0.0000%

--- I-SHAPED LAYOUT ---
Building dimensions: 529.5 x 132.4 m
Total area: 70100.0 sqm
Objective (total flow-distance): 17391.8
```

```
C:\Users\ruben\AppData\Local\Temp\ipykernel_7344\208478838.py:163: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colors = plt.cm.get_cmap('tab20', len(selected_depts))
```



In [ ]:
```python
# Print the quantified interaction values between departments
for dept_from in selected_depts:
    print(f"{dept_from}:")
    for dept_to in selected_depts:
        value = numeric_departments_matrix[dept_from][dept_to]
        print(f"  {dept_to}: {value}")
    print()
```

```
Inbound Dock:
  Inbound Dock: 0
  Outbound Staging: 0
  Packing / Wrap / Banding: 0
  Pallet Reserve Storage (Bulk): 0
  Receiving/Staging: 9
  Shipping Dock: 0
  Phantom: 0

Outbound Staging:
  Inbound Dock: 0
  Outbound Staging: 0
  Packing / Wrap / Banding: 9
  Pallet Reserve Storage (Bulk): 0
  Receiving/Staging: 0
  Shipping Dock: 9
  Phantom: 0

Packing / Wrap / Banding:
  Inbound Dock: 0
  Outbound Staging: 9
  Packing / Wrap / Banding: 0
  Pallet Reserve Storage (Bulk): 9
  Receiving/Staging: 0
  Shipping Dock: 0
  Phantom: 0

Pallet Reserve Storage (Bulk):
  Inbound Dock: 0
  Outbound Staging: 0
  Packing / Wrap / Banding: 9
  Pallet Reserve Storage (Bulk): 0
  Receiving/Staging: 4
  Shipping Dock: 0
  Phantom: 0

Receiving/Staging:
  Inbound Dock: 9
  Outbound Staging: 0
  Packing / Wrap / Banding: 0
  Pallet Reserve Storage (Bulk): 4
  Receiving/Staging: 0
  Shipping Dock: 0
  Phantom: 0

Shipping Dock:
  Inbound Dock: 0
  Outbound Staging: 9
  Packing / Wrap / Banding: 0
  Pallet Reserve Storage (Bulk): 0
  Receiving/Staging: 0
  Shipping Dock: 0
  Phantom: 0

Phantom:
  Inbound Dock: 0
  Outbound Staging: 0
  Packing / Wrap / Banding: 0
  Pallet Reserve Storage (Bulk): 0
  Receiving/Staging: 0
  Shipping Dock: 0
  Phantom: 0
```

Bonus Points

We have linearised the model from above to be able to solve the layout problem also for more departments.

In [ ]:
```python
import gurobipy as gp
from gurobipy import GRB
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math
```

```python
total_area = sum(departments[d] for d in selected_depts)

def solve_layout_linearized(layout_type, total_area, selected_depts, departments, numeric_departments_matrix,
    """
    Linear MILP version using:
    - Linearized Manhattan distances (alpha+, alpha-, beta+, beta-)
    - Perimeter-based area linearization with flexibility
    - Non-overlap constraints (Big-M)
    """

    model = gp.Model(f"linearized_layout_{layout_type}")
    model.Params.OutputFlag = 1

    BIG_M = 10000
    R = 3  # aspect ratio tolerance
    area_tolerance = area_flex  # e.g. 0.1 = 10%

    # Building fixed size (square)
    if layout_type == "I-shaped":
        B_side = math.sqrt(total_area)
        B_width = B_side * 2
        B_height = B_side * 0.5
    else:
        B_side = math.sqrt(total_area)
        B_width = B_side
        B_height = B_side

    # Add Phantom dept if needed
    if layout_type in ["L-shaped", "U-shaped"]:
        selected_depts = selected_depts.copy()
        if "Phantom" not in selected_depts:
            selected_depts.append("Phantom")
            departments["Phantom"] = total_area / 3
            numeric_departments_matrix["Phantom"] = {d: 0 for d in selected_depts}
            for d in selected_depts:
                numeric_departments_matrix[d]["Phantom"] = 0

    # Decision variables
    x_left, y_bottom, width, height = {}, {}, {}, {}
    alpha, beta = {}, {}
    z_x, z_y = {}, {}

    # Positive/negative parts for distance
    alpha_pos, alpha_neg, beta_pos, beta_neg = {}, {}, {}, {}

    for i in selected_depts:
        x_left[i] = model.addVar(lb=0, name=f"x_left_{i}")
        y_bottom[i] = model.addVar(lb=0, name=f"y_bottom_{i}")
        width[i] = model.addVar(lb=0, name=f"width_{i}")
        height[i] = model.addVar(lb=0, name=f"height_{i}")
        alpha[i] = model.addVar(lb=0, name=f"alpha_{i}")
        beta[i] = model.addVar(lb=0, name=f"beta_{i}")
    # Binary overlap indicators
    for i in selected_depts:
        z_x[i], z_y[i] = {}, {}
        for j in selected_depts:
            if i == j:
                continue
            z_x[i][j] = model.addVar(vtype=GRB.BINARY, name=f"z_x_{i}_{j}")
            z_y[i][j] = model.addVar(vtype=GRB.BINARY, name=f"z_y_{i}_{j}")

    # Distance positive/negative variables
    for i in selected_depts:
        alpha_pos[i], alpha_neg[i], beta_pos[i], beta_neg[i] = {}, {}, {}, {}
        for j in selected_depts:
            if i == j:
                continue
            alpha_pos[i][j] = model.addVar(lb=0, name=f"alpha_pos_{i}_{j}")
            alpha_neg[i][j] = model.addVar(lb=0, name=f"alpha_neg_{i}_{j}")
            beta_pos[i][j] = model.addVar(lb=0, name=f"beta_pos_{i}_{j}")
            beta_neg[i][j] = model.addVar(lb=0, name=f"beta_neg_{i}_{j}")

    model.update()

    # -----------------
```

```python
    # Core constraints
    # -----------------
    for i in selected_depts:
        sqrtA = math.sqrt(departments[i])

        # Aspect ratio bounds
        model.addConstr(width[i] >= sqrtA / R)
        model.addConstr(height[i] >= sqrtA / R)
        model.addConstr(width[i] <= sqrtA * R)
        model.addConstr(height[i] <= sqrtA * R)
        model.addConstr(x_left[i] + width[i] <= B_width)
        model.addConstr(y_bottom[i] + height[i] <= B_height)
        # Area linearization via perimeter
        # 4 * sqrt(A) = perimeter of square of area A
        per_square = 4 * sqrtA
        model.addConstr(
            2 * (width[i] + height[i]) >= per_square * (1 - area_tolerance),
            name=f"per_lb_{i}"
        )
        model.addConstr(
            2 * (width[i] + height[i]) <= per_square * (1 + area_tolerance),
            name=f"per_ub_{i}"
        )

        # Within building
        model.addConstr(x_left[i] + width[i] <= B_width)
        model.addConstr(y_bottom[i] + height[i] <= B_height)

        # Centroids
        model.addConstr(alpha[i] == x_left[i] + 0.5 * width[i])
        model.addConstr(beta[i] == y_bottom[i] + 0.5 * height[i])
            # Add these variables

    # Non-overlap
    for i in selected_depts:
        for j in selected_depts:
            if i == j:
                continue
            model.addConstr(x_left[j] >= x_left[i] + width[i] - BIG_M * (1 - z_x[i][j]))
            model.addConstr(y_bottom[j] >= y_bottom[i] + height[i] - BIG_M * (1 - z_y[i][j]))
            model.addConstr(z_x[i][j] + z_x[j][i] + z_y[i][j] + z_y[j][i] >= 1)

            # Linearized abs differences
            model.addConstr(alpha[i] - alpha[j] == alpha_pos[i][j] - alpha_neg[i][j])
            model.addConstr(beta[i] - beta[j] == beta_pos[i][j] - beta_neg[i][j])

    # Layout-type constraints
    if layout_type == "I-shaped":
        model.addConstr(x_left["Inbound Dock"] == 0)
        model.addConstr(x_left["Outbound Staging"] + width["Outbound Staging"] == B_width)

    elif layout_type == "L-shaped":
        # Bottom-left corner
        model.addConstr(x_left["Inbound Dock"] == 0)
        model.addConstr(y_bottom["Inbound Dock"] + height["Phantom"] == B_height)

        # Bottom-right corner
        model.addConstr(x_left["Outbound Staging"] + width["Outbound Staging"] == B_width)
        model.addConstr(y_bottom["Outbound Staging"] == 0)

        # Phantom (top-left corner)
        model.addConstr(x_left["Phantom"] == 0)
        model.addConstr(y_bottom["Phantom"] == 0)


    elif layout_type == "U-shaped":
        # Inbound dock (bottom-left)
        model.addConstr(x_left["Inbound Dock"] == 0)
        model.addConstr(y_bottom["Inbound Dock"] == 0)

        # Outbound staging (top-left)
        model.addConstr(x_left["Outbound Staging"] == 0)
        model.addConstr(y_bottom["Outbound Staging"] + height["Outbound Staging"] == B_height)

        # Phantom in middle
```

```python
        model.addConstr(x_left["Phantom"] == 0)
        #model.addConstr(x_left["Phantom"] + 0.5 * width["Phantom"] == B_width / 2)
        model.addConstr(y_bottom["Phantom"] + 0.5 * height["Phantom"] == B_height / 2)
        #model.addConstr(x_left["Phantom"] >= 0.1 * B_width)

    # Objective: minimize total flow * (|Δα| + |Δβ|) = α+ + α- + β+ + β-

    obj = gp.quicksum(
        numeric_departments_matrix[i][j] *
        (alpha_pos[i][j] + alpha_neg[i][j] + beta_pos[i][j] + beta_neg[i][j])
        for i in selected_depts for j in selected_depts if i != j
    )

    model.setObjective(obj, GRB.MINIMIZE)
    model.optimize()

    # Plot results (unchanged except width/height already consistent)
    if model.status in (GRB.OPTIMAL, GRB.TIME_LIMIT, GRB.SUBOPTIMAL):
        print(f"\n--- {layout_type.upper()} LAYOUT ---")
        print(f"Building dimensions: {B_width:.1f} x {B_height:.1f} m")
        print(f"Total area: {B_width * B_height:.1f} sqm")
        print(f"Objective (total flow-distance): {model.ObjVal:.1f}")

        fig, ax = plt.subplots(figsize=(15, 10))
        colors = plt.cm.get_cmap('tab20', len(selected_depts))

        # Warehouse (building) outline
        building_outline = patches.Rectangle(
            (0, 0),
            B_width,
            B_height,
            facecolor='none',
            edgecolor='red',
            linewidth=2,
            linestyle='--',
            label='Building'
        )
        ax.add_patch(building_outline)
        ax.text(B_width / 2, B_height + 0.02 * B_height,
                f"Warehouse: {B_width:.1f} x {B_height:.1f} m",
                ha='center', va='bottom', fontsize=12)

        for idx, i in enumerate(selected_depts):
            xl = x_left[i].X
            xr = xl + width[i].X
            yb = y_bottom[i].X
            yt = yb + height[i].X
            # Check if the department is Phantom
            if i == "Phantom":
                facecolor = 'white'
                label_text = ""  # no name
                show_marker = False
                show_area = False
            else:
                facecolor = colors(idx)
                label_text = i
                show_marker = True
                show_area = True

            rect = patches.Rectangle((xl, yb), xr - xl, yt - yb,
                                     edgecolor='black', facecolor=facecolor, alpha=0.7)
            ax.add_patch(rect)

            # Add department name
            if label_text:
                ax.text((xl + xr) / 2, (yb + yt) / 2, label_text, ha='center', va='center',
                        fontsize=10, fontweight='bold',
                        bbox=dict(facecolor='white', alpha=0.7, edgecolor='none'))

            # Add marker if not Phantom
            if show_marker:
                ax.plot((xl + xr) / 2, (yb + yt) / 2, 'kx', markersize=8)

            # Add area text if not Phantom
            if show_area:
```

```python
                ax.text(xl + 5, yt - 5, f"{(xr - xl) * (yt - yb):.0f} sqm", fontsize=8)

        ax.set_xlim(0, B_width)
        ax.set_ylim(0, B_height)
        ax.set_aspect('equal')
        ax.set_title(f"{layout_type} Layout\nTotal Flow-Distance: {model.ObjVal:.1f}", fontsize=14)
        ax.grid(True, linestyle='--', alpha=0.6)
        ax.set_xlabel("Width (m)")
        ax.set_ylabel("Height (m)")
        ax.legend(loc='upper right')
        plt.show()
    else:
        print(f"No feasible solution for {layout_type}, status: {model.status}")
```

In [41]:

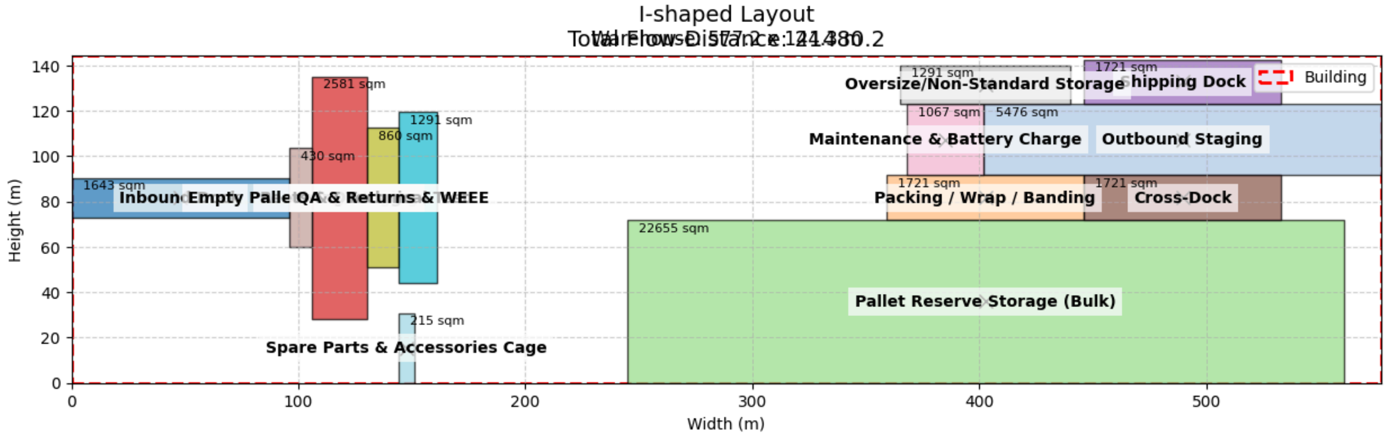
```python
# Run for all layouts
for layout_type in ["I-shaped"]:
    selected_depts = load_all_data()
    solve_layout_linearized(layout_type, total_area=total_area, selected_depts=selected_depts, departments=de
```

```
Set parameter OutputFlag to value 1
```

```
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (win64 - Windows 11.0 (26100.2))

CPU model: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Optimize a model with 938 rows, 1014 columns and 3409 nonzeros
Model fingerprint: 0xb14be047
Variable types: 702 continuous, 312 integer (312 binary)
Coefficient statistics:
  Matrix range     [5e-01, 1e+04]
  Objective range  [1e+00, 2e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
Presolve removed 493 rows and 616 columns
Presolve time: 0.01s
Presolved: 445 rows, 398 columns, 1727 nonzeros
Variable types: 112 continuous, 286 integer (286 binary)
Found heuristic solution: objective 47694.068825
Found heuristic solution: objective 44413.082917
Found heuristic solution: objective 33377.826575

Root relaxation: objective 1.429315e+04, 209 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 14293.1519    0   64 33377.8266 14293.1519  57.2%     -    0s
H    0     0                    33090.391101 14293.1519  56.8%     -    0s
     0     0 14877.8900    0   66 33090.3911 14877.8900  55.0%     -    0s
H    0     0                    31791.107725 15171.0426  52.3%     -    0s
     0     0 15171.0426    0   65 31791.1077 15171.0426  52.3%     -    0s
     0     0 18167.8280    0   68 31791.1077 18167.8280  42.9%     -    0s
H    0     0                    27846.372533 18167.8280  34.8%     -    0s
H    0     0                    24597.154267 18182.7850  26.1%     -    0s
     0     0 18182.7850    0   62 24597.1543 18182.7850  26.1%     -    0s
     0     0 18182.7850    0   62 24597.1543 18182.7850  26.1%     -    0s
     0     0 18668.9822    0   62 24597.1543 18668.9822  24.1%     -    0s
     0     0 18702.8754    0   58 24597.1543 18702.8754  24.0%     -    0s
     0     0 18717.4480    0   63 24597.1543 18717.4480  23.9%     -    0s
     0     0 18717.4480    0   64 24597.1543 18717.4480  23.9%     -    0s
     0     0 18790.1271    0   51 24597.1543 18790.1271  23.6%     -    0s
     0     0 18790.1271    0   51 24597.1543 18790.1271  23.6%     -    0s
     0     0 18790.1271    0   55 24597.1543 18790.1271  23.6%     -    0s
H    0     0                    24316.776863 18790.1271  22.7%     -    0s
     0     0 18790.1271    0   54 24316.7769 18790.1271  22.7%     -    0s
     0     0 18790.1932    0   38 24316.7769 18790.1932  22.7%     -    0s
     0     0 18793.0369    0   38 24316.7769 18793.0369  22.7%     -    0s
     0     0 18793.0369    0   45 24316.7769 18793.0369  22.7%     -    0s
     0     0 18793.0369    0   33 24316.7769 18793.0369  22.7%     -    0s
     0     2 18793.0369    0   31 24316.7769 18793.0369  22.7%     -    0s
H  108   132                    24198.117687 18863.4093  22.0%  10.6    0s
H  163   185                    23301.381467 18863.4093  19.0%   9.9    0s
H  187   185                    22644.314892 18863.4093  16.7%   9.5    0s
H  252   256                    22363.904646 18872.8662  15.6%   8.8    0s
H  260   256                    21578.863013 18872.8662  12.5%   8.7    0s
H 1516   718                    21535.430741 19397.1399  9.93%   8.4    0s
* 6387   688              34    21526.580182 20990.6550  2.49%   9.2    1s
* 7466   178              26    21480.191909 21283.5230  0.92%   9.2    1s

Cutting planes:
  Gomory: 25
  Cover: 3
  Implied bound: 39
  MIR: 152
  Inf proof: 14
  RLT: 11
  Relax-and-lift: 2

Explored 7978 nodes (73505 simplex iterations) in 1.33 seconds (0.81 work units)
Thread count was 12 (of 12 available processors)

Solution count 10: 21480.2 21526.6 21535.4 ... 24597.2

Optimal solution found (tolerance 1.00e-04)
Best objective 2.148019190917e+04, best bound 2.148019190917e+04, gap 0.0000%
```

```
--- I-SHAPED LAYOUT ---
Building dimensions: 577.2 x 144.3 m
Total area: 83300.0 sqm
Objective (total flow-distance): 21480.2
```

```
C:\Users\chant\AppData\Local\Temp\ipykernel_34796\3827421511.py:181: MatplotlibDeprecationWarning: The get_cm
ap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]``
or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colors = plt.cm.get_cmap('tab20', len(selected_depts))
```

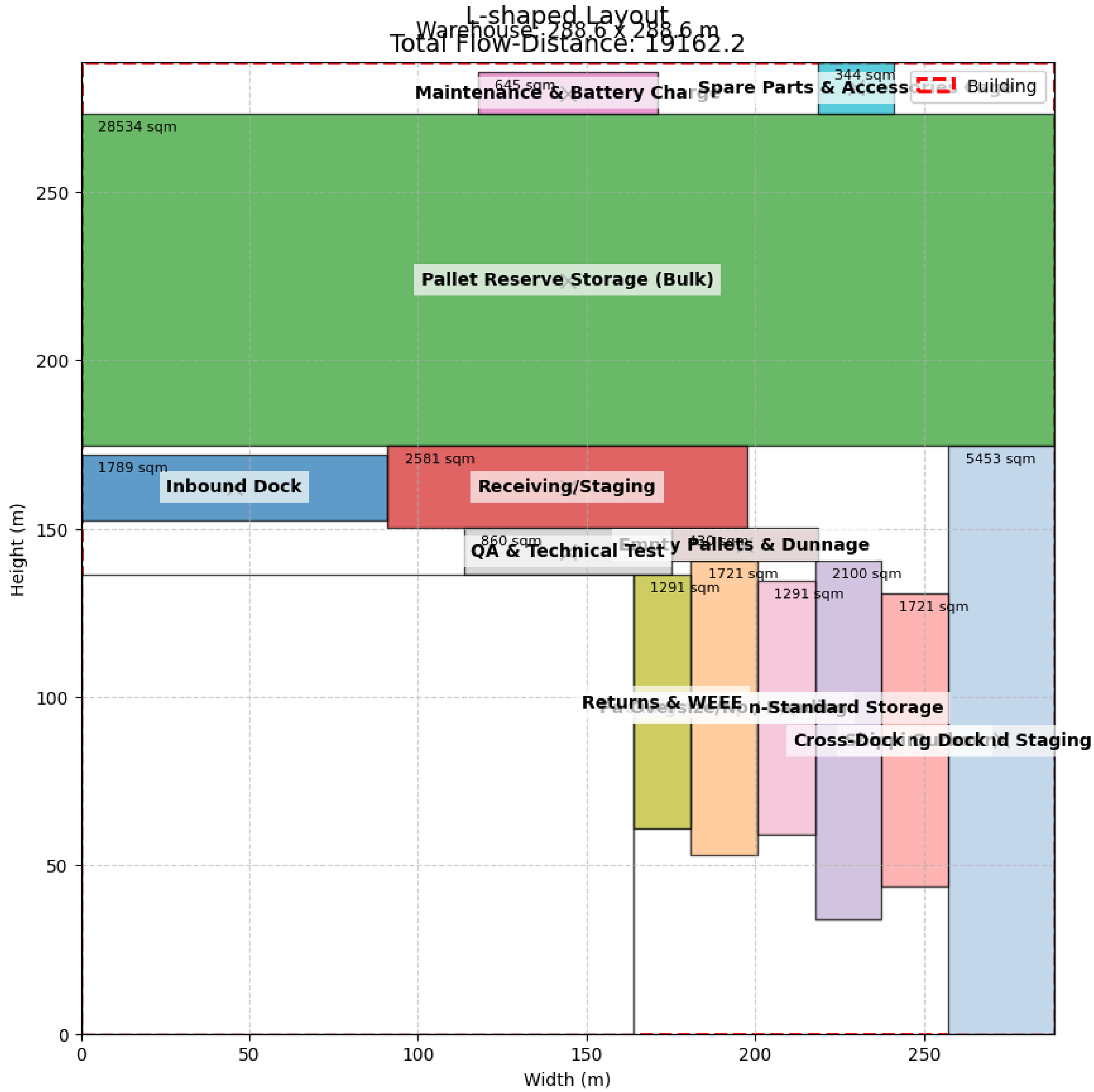


In [42]:
```python
# Run for all layouts
for layout_type in ["L-shaped"]:
    selected_depts = load_all_data()
    solve_layout_linearized(layout_type, total_area=total_area, selected_depts=selected_depts, departments=de
```

```
Set parameter OutputFlag to value 1
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (win64 - Windows 11.0 (26100.2))

CPU model: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Optimize a model with 1084 rows, 1176 columns and 3956 nonzeros
Model fingerprint: 0xb19da74b
Variable types: 812 continuous, 364 integer (364 binary)
Coefficient statistics:
  Matrix range     [5e-01, 1e+04]
  Objective range  [1e+00, 2e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
Presolve removed 639 rows and 790 columns
Presolve time: 0.01s
Presolved: 445 rows, 386 columns, 1710 nonzeros
Variable types: 111 continuous, 275 integer (275 binary)

Root relaxation: objective 1.144759e+04, 221 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 11447.5897    0   42          - 11447.5897      -     -    0s
H    0     0                    38922.574346 11447.5897  70.6%     -    0s
H    0     0                    24408.551107 11447.5897  53.1%     -    0s
     0     0 12116.4446    0   70 24408.5511 12116.4446  50.4%     -    0s
     0     0 12401.0890    0   77 24408.5511 12401.0890  49.2%     -    0s
     0     0 12488.7883    0   76 24408.5511 12488.7883  48.8%     -    0s
     0     0 12488.7883    0   76 24408.5511 12488.7883  48.8%     -    0s
     0     0 13325.5529    0   79 24408.5511 13325.5529  45.4%     -    0s
     0     0 13480.5036    0   71 24408.5511 13480.5036  44.8%     -    0s
     0     0 13480.5036    0   76 24408.5511 13480.5036  44.8%     -    0s
     0     0 13480.5036    0   79 24408.5511 13480.5036  44.8%     -    0s
     0     0 14043.1543    0   76 24408.5511 14043.1543  42.5%     -    0s
     0     0 14358.2859    0   70 24408.5511 14358.2859  41.2%     -    0s
     0     0 14367.5688    0   66 24408.5511 14367.5688  41.1%     -    0s
     0     0 14367.7141    0   71 24408.5511 14367.7141  41.1%     -    0s
H    0     0                    24002.774975 14451.0252  39.8%     -    0s
     0     0 14451.0252    0   65 24002.7750 14451.0252  39.8%     -    0s
     0     0 14526.5895    0   66 24002.7750 14526.5895  39.5%     -    0s
     0     0 14526.5895    0   65 24002.7750 14526.5895  39.5%     -    0s
H    0     0                    23535.086296 14526.5895  38.3%     -    0s
     0     0 14526.5895    0   71 23535.0863 14526.5895  38.3%     -    0s
     0     0 14554.1729    0   71 23535.0863 14554.1729  38.2%     -    0s
H    0     0                    23192.092900 14554.1729  37.2%     -    0s
     0     0 14554.1729    0   75 23192.0929 14554.1729  37.2%     -    0s
     0     0 14608.0589    0   81 23192.0929 14608.0589  37.0%     -    0s
H    0     0                    22925.209255 14608.0589  36.3%     -    0s
     0     0 14608.0589    0   81 22925.2093 14608.0589  36.3%     -    0s
     0     0 15788.9562    0   73 22925.2093 15788.9562  31.1%     -    0s
     0     2 15788.9562    0   57 22925.2093 15788.9562  31.1%     -    0s
H   43    72                    21659.000654 16192.8416  25.2%  22.7    0s
H  103   120                    21215.691618 16192.8416  23.7%  13.3    0s
*  563   371              59    20138.944803 16701.5465  17.1%  11.1    0s
H 1592   663                    19865.930754 17232.7060  13.3%  11.3    0s
H 2279   818                    19864.333007 17380.4158  12.5%  13.5    1s
H 2429   772                    19696.962293 17509.8166  11.1%  13.3    1s
H 3305   756                    19669.963189 17865.0358  9.18%  13.0    1s
* 3523   682              63    19633.913013 17928.4512  8.69%  12.9    1s
H 3529   646                    19533.952390 17928.4512  8.22%  12.9    1s
H 3873   619                    19388.321115 18102.6204  6.63%  12.6    1s
* 4306   775              56    19368.763313 18162.4102  6.23%  12.2    2s
H 5522  1081                    19367.025690 18322.5966  5.39%  11.5    2s
* 9852  1826              60    19348.401721 18670.3417  3.50%  10.4    2s
*11586  1771              72    19274.432198 18783.5237  2.55%  10.2    3s
H14803  1446                    19223.069785 18985.7810  1.23%   9.9    3s
H14880  1424                    19219.640828 18985.9561  1.22%   9.9    3s
*15251  1460              66    19217.799987 18985.9561  1.21%   9.8    3s
*15651  1407              57    19213.431742 19006.4588  1.08%   9.7    3s
*15840  1278              66    19197.048329 19014.0516  0.95%   9.7    3s
*17057  1080              68    19190.149573 19043.7312  0.76%   9.6    3s
*17460   938              60    19170.361360 19044.5644  0.66%   9.6    3s
*17883   629              56    19163.744102 19069.2083  0.49%   9.5    3s
```

```
*17884   608              56    19162.169654 19069.2083  0.49%   9.5    3s

Cutting planes:
  Gomory: 36
  Cover: 16
  Implied bound: 20
  MIR: 103
  Flow cover: 224
  Inf proof: 8
  RLT: 7
  Relax-and-lift: 4

Explored 19192 nodes (184503 simplex iterations) in 4.17 seconds (1.68 work units)
Thread count was 12 (of 12 available processors)

Solution count 10: 19162.2 19163.7 19170.4 ... 19274.4

Optimal solution found (tolerance 1.00e-04)
Best objective 1.916216965376e+04, best bound 1.916216965376e+04, gap 0.0000%

--- L-SHAPED LAYOUT ---
Building dimensions: 288.6 x 288.6 m
Total area: 83300.0 sqm
Objective (total flow-distance): 19162.2
```

```
C:\Users\chant\AppData\Local\Temp\ipykernel_34796\3827421511.py:181: MatplotlibDeprecationWarning: The get_cm
ap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]``
or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colors = plt.cm.get_cmap('tab20', len(selected_depts))
```

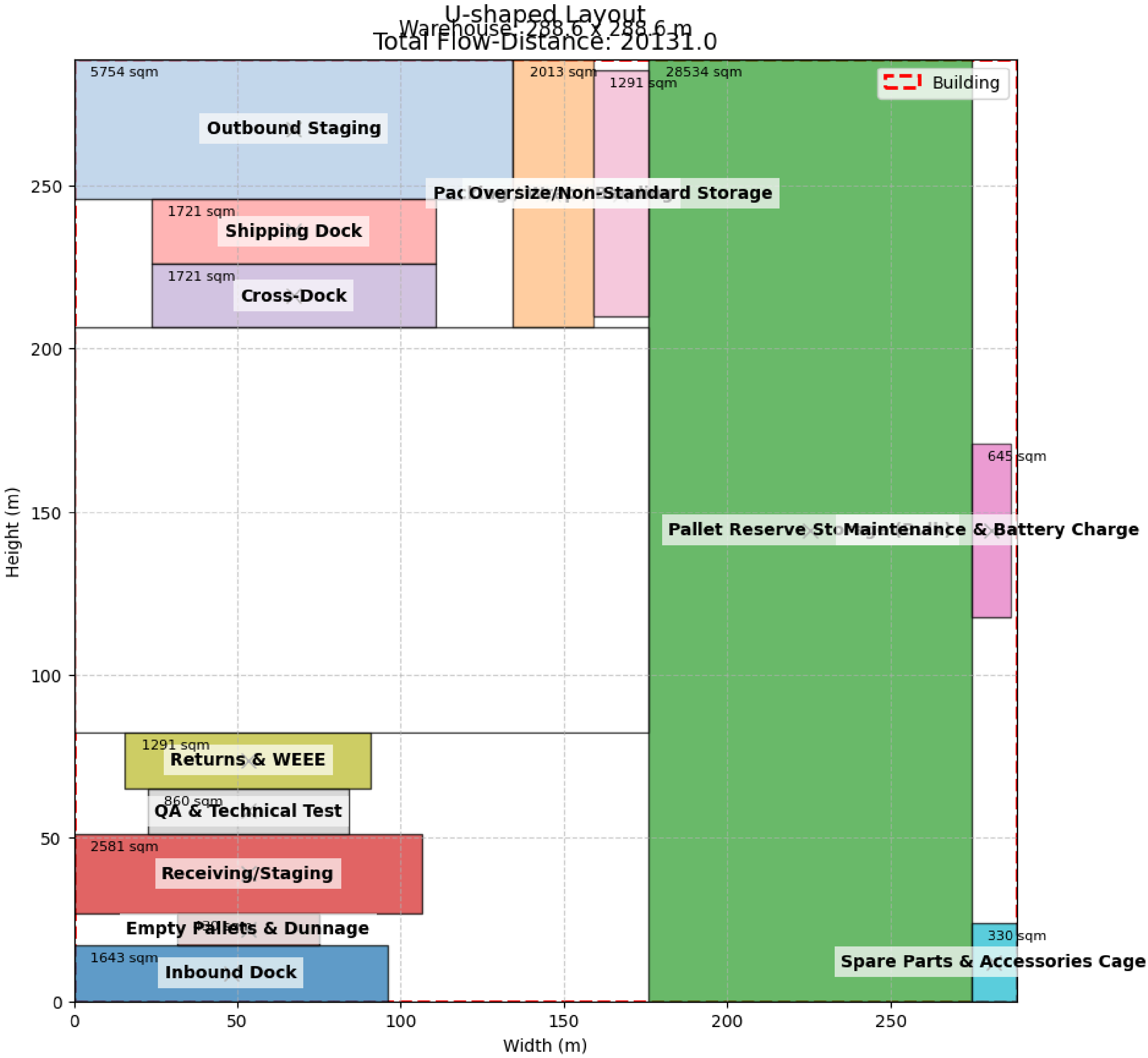



```
In [55]: # Run for all layouts
for layout_type in ["U-shaped"]:
    selected_depts = load_all_data()
    solve_layout_linearized(layout_type, total_area=total_area, selected_depts=selected_depts, departments=de
```

```
Set parameter OutputFlag to value 1
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (win64 - Windows 11.0 (26100.2))

CPU model: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Optimize a model with 1084 rows, 1176 columns and 3956 nonzeros
Model fingerprint: 0xa8258a50
Variable types: 812 continuous, 364 integer (364 binary)
Coefficient statistics:
  Matrix range     [5e-01, 1e+04]
  Objective range  [1e+00, 2e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
Presolve removed 641 rows and 793 columns
Presolve time: 0.01s
Presolved: 443 rows, 383 columns, 1703 nonzeros
Variable types: 110 continuous, 273 integer (273 binary)

Root relaxation: objective 1.269845e+04, 211 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 12698.4473    0   50          - 12698.4473      -     -    0s
H    0     0                    36115.709994 12698.4473  64.8%     -    0s
H    0     0                    33054.015327 12698.4473  61.6%     -    0s
     0     0 13508.6259    0   56 33054.0153 13508.6259  59.1%     -    0s
H    0     0                    32765.997952 13627.3718  58.4%     -    0s
     0     0 14070.7933    0   56 32765.9980 14070.7933  57.1%     -    0s
     0     0 14113.6715    0   56 32765.9980 14113.6715  56.9%     -    0s
H    0     0                    32693.334344 14227.1405  56.5%     -    0s
     0     0 14227.1405    0   62 32693.3343 14227.1405  56.5%     -    0s
     0     0 14227.1405    0   60 32693.3343 14227.1405  56.5%     -    0s
     0     0 14248.5434    0   60 32693.3343 14248.5434  56.4%     -    0s
     0     0 14257.8473    0   61 32693.3343 14257.8473  56.4%     -    0s
H    0     0                    23948.097188 14459.7862  39.6%     -    0s
     0     0 14736.1076    0   69 23948.0972 14736.1076  38.5%     -    0s
     0     0 14772.2655    0   69 23948.0972 14772.2655  38.3%     -    0s
     0     0 14774.6876    0   66 23948.0972 14774.6876  38.3%     -    0s
     0     0 14777.1066    0   71 23948.0972 14777.1066  38.3%     -    0s
     0     0 14777.1066    0   72 23948.0972 14777.1066  38.3%     -    0s
     0     0 15102.1981    0   65 23948.0972 15102.1981  36.9%     -    0s
H    0     0                    23868.991071 15108.9103  36.7%     -    0s
     0     0 15146.6685    0   71 23868.9911 15146.6685  36.5%     -    0s
     0     0 15146.6685    0   71 23868.9911 15146.6685  36.5%     -    0s
     0     0 15234.6326    0   69 23868.9911 15234.6326  36.2%     -    0s
     0     0 15289.6386    0   75 23868.9911 15289.6386  35.9%     -    0s
     0     0 15289.6386    0   69 23868.9911 15289.6386  35.9%     -    0s
H    0     0                    22612.774473 15289.6386  32.4%     -    0s
     0     0 15289.6386    0   68 22612.7745 15289.6386  32.4%     -    0s
     0     0 15289.6386    0   71 22612.7745 15289.6386  32.4%     -    0s
     0     0 15314.3213    0   76 22612.7745 15314.3213  32.3%     -    0s
     0     0 15320.5622    0   77 22612.7745 15320.5622  32.2%     -    0s
H    0     0                    22023.870582 15337.3204  30.4%     -    0s
     0     0 15337.3204    0   71 22023.8706 15337.3204  30.4%     -    0s
     0     0 15339.1866    0   75 22023.8706 15339.1866  30.4%     -    0s
     0     0 15339.1866    0   74 22023.8706 15339.1866  30.4%     -    0s
H    0     0                    21862.675882 15507.3770  29.1%     -    0s
     0     0 15507.3770    0   74 21862.6759 15507.3770  29.1%     -    0s
     0     0 15524.8139    0   73 21862.6759 15524.8139  29.0%     -    0s
     0     0 15526.6923    0   75 21862.6759 15526.6923  29.0%     -    0s
     0     0 15527.7169    0   79 21862.6759 15527.7169  29.0%     -    0s
     0     0 15537.8544    0   77 21862.6759 15537.8544  28.9%     -    0s
     0     0 15541.0576    0   73 21862.6759 15541.0576  28.9%     -    0s
     0     0 15541.0576    0   76 21862.6759 15541.0576  28.9%     -    0s
     0     0 15568.4874    0   81 21862.6759 15568.4874  28.8%     -    0s
     0     0 15601.9397    0   86 21862.6759 15601.9397  28.6%     -    0s
     0     0 15601.9397    0   88 21862.6759 15601.9397  28.6%     -    0s
H    0     0                    21170.559874 15601.9397  26.3%     -    0s
     0     0 15627.7607    0   73 21170.5599 15627.7607  26.2%     -    0s
     0     0 15628.9046    0   79 21170.5599 15628.9046  26.2%     -    0s
     0     0 15655.9546    0   77 21170.5599 15655.9546  26.0%     -    0s
     0     0 15661.7638    0   76 21170.5599 15661.7638  26.0%     -    0s
     0     0 15662.1845    0   77 21170.5599 15662.1845  26.0%     -    0s
```

```
     0     0 15662.1845    0   74 21170.5599 15662.1845  26.0%     -    0s
     0     0 15662.2324    0   63 21170.5599 15662.2324  26.0%     -    0s
H    0     0                    20499.625981 15662.2324  23.6%     -    0s
     0     2 15662.2324    0   62 20499.6260 15662.2324  23.6%     -    0s
H   63    55                    20335.416413 16252.6633  20.1%  25.1    0s
H 1212   457                    20296.842089 18053.5048  11.1%  11.6    0s
H 2670   738                    20251.402164 18704.8838  7.64%  11.1    0s
* 3309   801              35    20243.888868 18905.9907  6.61%  11.0    0s
H 3628   824                    20186.268143 18964.4677  6.05%  10.8    1s
H 4530   711                    20131.029311 19191.5183  4.67%  10.8    1s

Cutting planes:
  Gomory: 7
  Cover: 14
  Implied bound: 60
  Clique: 2
  MIR: 271
  Inf proof: 17
  RLT: 40
  Relax-and-lift: 11

Explored 7836 nodes (84074 simplex iterations) in 1.69 seconds (1.09 work units)
Thread count was 12 (of 12 available processors)

Solution count 10: 20131 20186.3 20243.9 ... 22023.9

Optimal solution found (tolerance 1.00e-04)
Best objective 2.013102931136e+04, best bound 2.013102931136e+04, gap 0.0000%

--- U-SHAPED LAYOUT ---
Building dimensions: 288.6 x 288.6 m
Total area: 83300.0 sqm
Objective (total flow-distance): 20131.0
```

```
C:\Users\chant\AppData\Local\Temp\ipykernel_34796\2685057132.py:184: MatplotlibDeprecationWarning: The get_cm
ap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]``
or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colors = plt.cm.get_cmap('tab20', len(selected_depts))
```


U-shaped Layout
Warehouse: 288.6 x 288.6 m
Total Flow-Distance: 20131.0
Building
Outbound Staging
5754 sqm
2013 sqm
1291 sqm
28534 sqm
Oversize/Non-Standard Storage
Shipping Dock
1721 sqm
Cross-Dock
1721 sqm
Pallet Reserve St
Maintenance & Battery Charge
645 sqm
Returns & WEEE
1291 sqm
QA & Technical Test
860 sqm
Receiving/Staging
2581 sqm
Empty Pallets & Dunnage
Inbound Dock
1643 sqm
Spare Parts & Accessories Cage
330 sqm
Height (m)
Width (m)
0
50
100
150
200
250