

Tutorial_1_AIE_1901_LLM_for_Optimization

September 10, 2025

0.1 Upload the Solver

```
[1]: from copty import COPT
```

```
import pandas as pd
from copty import *

# Create COPT environment
env = Env()

# === Create COPT model ===
model = env.createModel("m")
```

Cardinal Optimizer v7.2.11. Build date Aug 1 2025
Copyright Cardinal Operations 2025. All Rights Reserved

0.2 Parameter

```
[2]: T = 30 ## Finish within 30 days
N = 64 ## we have 64 locations, but we don't need to visit them all
K = 2 ## Only two kinds of resources: food and water
M = 1000 # a large number
L = 1200 # the weight restriction
J = 10000 # we have 10,000 dollars at the beginning
W = [0, 3, 2] # weight for resource W[1] == water; W[2] == food
P = [0, 5, 10] # price for resource P[1] == water; P[2] == food

comsumption = [{"": 5, "": 8, "": 10}, {"": 7, "": 6, "": 10}]
## comsumption[1] == water; comsumption[2] == food
```

```
[3]: # t -- day, i -- location, k -- resource
t_i = [(t, i) for t in range(1, T + 1) for i in range(1, N + 1)] # 1 ~ T, 1 ~ N
t_k = [(t, k) for t in range(1, T + 1) for k in range(1, K + 1)] # 1 ~ T, 1 ~ K
i_k = [(i, k) for i in range(1, N + 1) for k in range(1, K + 1)] # 1 ~ N, 1 ~ K
t_0_i = [(t, i) for t in range(0, T + 1) for i in range(1, N + 1)] # 0 ~ T, 1 ~ N
```

```

t_0_k = [(t, k) for t in range(0, T + 1) for k in range(1, K + 1)] # 0 ~ T, 1 ~ K
t_i_k = [(t, i, k) for t in range(1, T + 1) for i in range(1, N + 1) for k in range(1, K + 1)]
# 1 ~ T, 1 ~ N, 1 ~ K

## k=1: water; k=2: food

```

[4]: import csv

```

def calculate_adjacent_hexagons():
    adjacent_map = {}
    for i in range(64): # 0-63
        row = i // 8
        is_even_row = row % 2 == 0
        adjacent = []

        adjacent.append(i) # 1

        #
        if i % 8 != 0:
            adjacent.append(i - 1)
        if i % 8 != 7:
            adjacent.append(i + 1)

        #
        if row > 0:
            if is_even_row:
                if i % 8 != 0:
                    adjacent.append(i - 9)
                    adjacent.append(i - 8)
            else:
                adjacent.append(i - 8)
                if i % 8 != 7:
                    adjacent.append(i - 7)

        #
        if row < 7:
            if is_even_row:
                adjacent.append(i + 8)
                if i % 8 != 0:
                    adjacent.append(i + 7)
            else:
                if i % 8 != 7:
                    adjacent.append(i + 9)
                    adjacent.append(i + 8)

```

```

#      0-63
adjacent = list(set([x for x in adjacent if 0 <= x < 64]))
adjacent_map[i] = adjacent
return adjacent_map

#      1
def generate_adjacency_matrix():
    adjacent_map = calculate_adjacent_hexagons()
    matrix = []

#      1-64
header = [''] + [i + 1 for i in range(64)]
matrix.append(header)

#      1-64  0 1
for i in range(64):
    row_data = [0] * 64
    for neighbor in adjacent_map[i]:
        row_data[neighbor] = 1
    #      i+1
    matrix_row = [i + 1] + row_data
    matrix.append(matrix_row)

return matrix

#      CSV  GBK
def save_to_csv(matrix, file_path='adjacency_matrix.csv'):
    with open(file_path, 'w', newline='', encoding='gbk') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerows(matrix)

if __name__ == "__main__":
    adjacency_matrix = generate_adjacency_matrix()
    save_to_csv(adjacency_matrix)

```

```

[5]: import pandas as pd

# Weather data for 30 days
weather_conditions = [
    ['', '', '', '', '', '', '', '', '', '', '', '',
     '', '', '', '', '', '', '', '', '', '',
     '', '', '', '', '', '', '', '', '', '',
     ''],
]

# Create DataFrame with days as columns and weather as the single row
# This structure allows weather_data.loc[0, str(t)] to work

```

```

df = pd.DataFrame([weather_conditions], columns=[str(i) for i in range(1, 31)])

# Save to CSV
df.to_csv('weather.csv', index=False, encoding='utf-8-sig')

```

[6]:

```

weather_data = pd.read_csv('weather.csv', encoding='utf-8')
pos_data = pd.read_csv('adjacency_matrix.csv', encoding='gbk')

# Special Location
village = {39: [1, 2], 62: [1, 2]} ## you can purchase the resource at village
mine_pos = {30: 1000, 55: 1000}
end_pos = [64]

```

[7]:

```

A = {(t, k): consumption[k][weather_data.loc[0, str(t)]] for t, k in t_k}
# base consumption for resource k on day t
B = {t: weather_data.loc[0, str(t)] == " " for t in range(1, T + 1)}
# sandstorm weather
D = {(i, k): int(i in village.keys() and k in village[i]) for i, k in i_k}
# binary - whether it is available to buy resource k at location i
E = {i: int(i in mine_pos) for i in range(1, N + 1)}
# binary - whether location i have minery
F = {i: int(i in end_pos) for i in range(1, N + 1)}
# binary - whether location i is the destination
G = {i: mine_pos[i] if i in mine_pos else 0 for i in range(1, N + 1)}
# integer - the mining benefit at location i (=1000 if E[i]==1)
H = {(i, j): pos_data.loc[i - 1, str(j)] if i != j else 1 for i in range(1, N + 1) for j in range(1, N + 1)}
# binary - whether location i and j are neighbour

```

0.3 Decision Variable

[8]:

```

# Binary variable - whether we arrive at location i at day t
x = model.addVars(t_0_i, vtype=COPT.BINARY, nameprefix='x')

# Binary variable - whether the player stay at location i at day t
y = model.addVars(t_i, vtype=COPT.BINARY, nameprefix='y')

# Integer variable - the quantity of the purchased resource k at day t
z = model.addVars(t_0_k, vtype=COPT.INTEGER, nameprefix='z', lb=0)

# Binary variable - whether the player dig mine at day t
w = model.addVars([i for i in range(1, T + 1)], vtype=COPT.BINARY, nameprefix='w')

```

[9]:

```

# Binary variable - whether arrive at location with resource k at day t
a = model.addVars(t_0_k, vtype=COPT.BINARY, nameprefix='a')

```

```

model.addConstrs((a[(t, k)] == quicksum(x[(t, i)] * D[(i, k)] for i in range(1, u
    ↪N + 1)) for t, k in t_0_k),
                  r"\"a_{t,k}=\sum_{i=1}^{N}D_{i,k}"")"

# Binary variable - whether available to dig minery
b = model.addVars([i for i in range(1, T + 1)], vtype=COPT.BINARY, u
    ↪nameprefix='b')
model.addConstrs((b[t] == quicksum(y[(t, i)] * E[i] for i in range(1, N + 1))u
    ↪for t in range(1, T + 1)),
                  r"\"b_t = \sum_{i=1}^{N}y_{t,i}E_i"")"

# Binary variable - whether the player arrived at destination
d = model.addVars([i for i in range(0, T + 1)], vtype=COPT.BINARY, u
    ↪nameprefix='d')
model.addConstrs((d[t] == quicksum(x[(t, i)] * F[i] for i in range(1, N + 1))u
    ↪for t in range(0, T + 1)),
                  r"\"b_t = \sum_{i=1}^{N}x_{t,i}F_i"")"

# Continuous variable - the quantity of resource k at day t before consumption
u = model.addVars(t_0_k, vtype=COPT.INTEGER, nameprefix='u', lb=0)

# Continuous variable - the quantity of resource k at day t after consumption
v = model.addVars(t_0_k, vtype=COPT.INTEGER, nameprefix='v', lb=0)

# Continuous variable - the left fund at day t
s = model.addVars([i for i in range(0, T + 1)], vtype=COPT.INTEGER, u
    ↪nameprefix='s')

```

0.4 Constraints

```
[10]: # (1) Starting point
model.addConstr((x[(0, 1)] == 1), "1 x_{0,1} = 1")

# (2) Must get back to destination before Day 30
model.addConstr((x[(T, N)] == 1), "2 x_{T,N} = 1")

# (3) Must present at one location every day
model.addConstrs((x.sum(t, "*") == 1 for t in range(0, T + 1)), nameprefix=r"3"u
    ↪\sum_{i=1}^{N}x_{t,i} = 1")
# x size = [(t, i) for t in range(0, T + 1) for i in range(1, N + 1)]
# for t in range(0, T + 1):
#     model.addConstr(x.sum(t, "*") == 1)

# (4) Can only visit adjacent regions
model.addConstrs(((x[(t, i)] + x[(t - 1, j)] <= H[(i, j)] + 1) for t, i in t_iu
    ↪for j in range(1, N + 1)),
                  nameprefix="4 x_{t,i} + x_{t-1,i} <= H_{i,j} + 1")
```

```

# (5) Weight limitation
model.addConstrs((quicksum(v[(t, k)] * W[k] for k in range(1, K + 1)) <= L for
    ↪ t in range(0, T + 1)),
    nameprefix=r"6 \sum_{k=1}^{K} v_{t,k}*W_k <= L")

# (6) Stay at original place when sandstorm
model.addConstrs((y.sum(t, "*") == 1 for t in range(1, T + 1) if B[t]),
    ↪ nameprefix="7 stop")

# (7) Dig mine
model.addConstrs((w[t] <= b[t] for t in range(1, T + 1)), nameprefix="8 w_t <=
    ↪ b_t")

# (8) Resource Purchase
model.addConstrs((z[(t, k)] <= a[(t, k)] * M for t in range(1, T + 1) for k in
    ↪ range(1, K + 1)),
    nameprefix="9 z_{t,k} <= a_{t,k}*M")

# (9) No return after arrive at destination
model.addConstrs((d[t] <= d[t + 1] for t in range(0, T)), nameprefix="10"
    ↪ d[t]<=d[t+1"])

# (10) Stay and moving constraints
model.addConstrs((y[(t, i)] <= x[(t, i)] for t, i in t_i), nameprefix="11.1"
    ↪ y_{t,i} <= x_{t,i}")
model.addConstrs((y[(t, i)] <= x[(t - 1, i)] for t, i in t_i), nameprefix="11.2"
    ↪ y_{t,i} <= x_{t-1,i}")
model.addConstrs((x[(t - 1, i)] + x[(t, i)] <= y[(t, i)] + 1 for t, i in t_i),
    nameprefix="11.3 x_{t-1,i} + x_{t,i} <= y_{t,i} + 1")

```

[10]: <coptcore.tupledict at 0x13a6c7c10>

```

[11]: #
# (1)
model.addConstrs((u[(t, k)] == v[(t - 1, k)] - \
    (2 * w[t] - y.sum(t, "*") + 2 - d[t - 1]) * A[(t, k)] \
    for t in range(1, T + 1) for k in range(1, K + 1)),
    nameprefix=r"u_{t,k} = v_{t-1,k} - (2*w_t-\sum_{i=1}^N y_{t,i})" \
    ↪ + 2 - d_{t-1}) "
    r"*\sum_{i=1}^N A_{t,i,k}*x_{t-1,i}")
model.addConstrs((v[(t, k)] == u[(t, k)] + z[(t, k)] for t, k in t_0_k),
    ↪ nameprefix='v_{t,k} = u_{t,k} + z_{t,k}')

# (2) 0

```

```

model.addConstrs((u[(0, k)] == 0 for k in range(1, K + 1)),  

    ↪nameprefix="u_{0,k}=0")

# (3)
model.addConstr((s[0] == J - quicksum(z[(0, k)] * P[k] for k in range(1, K + 1))),  

    ↪  

    r"s_{0}=J-\sum_{k=1}^{K}z_{0,k}*P_k")

# (4)
model.addConstrs((s[t] == s[t - 1] + 1000 * w[t] - 2 * quicksum(z[(t, k)] *  

    ↪P[k] for k in range(1, K + 1))  

    for t in range(1, T + 1)), nameprefix=r"s_{t} = s_{t-1} +"  

    ↪G*w_t - 2*\sum_{k=1}^{K}z_{t,k}*P_k")

```

[11]: <coptcore.tupledict at 0x13a6eab10>

0.5 Objective Function

[12]: model.setObjective(s[T] + 0.5 * quicksum(v[(T, k)] * P[k] for k in range(1, K + 1)), COPT.MAXIMIZE)

0.6 Optimal Solution

[13]: model.solve()

Model fingerprint: c6039872

Using Cardinal Optimizer v7.2.11 on macOS (aarch64)
Hardware has 11 cores and 11 threads. Using instruction set ARMV8 (30)
Maximizing a MIP problem

The original problem has:

129108 rows, 4274 columns and 266631 non-zero elements
4057 binaries and 217 integers

Starting the MIP solver with 11 threads and 32 tasks

Presolving the problem

The presolved problem has:

3384 rows, 1759 columns and 47548 non-zero elements
1651 binaries and 108 integers

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
0	1	--	0	3.284000e+04	--	Inf	0.99s
0	1	--	325	1.698776e+04	--	Inf	1.19s

0	1	--	496	1.669939e+04	--	Inf	1.29s
0	1	--	448	1.654993e+04	--	Inf	1.38s
0	1	--	380	1.645353e+04	--	Inf	1.43s
0	1	--	506	1.633498e+04	--	Inf	1.48s
0	1	--	529	1.619317e+04	--	Inf	1.56s
0	1	--	633	1.606137e+04	--	Inf	1.66s
0	1	--	566	1.600933e+04	--	Inf	1.76s
0	1	--	519	1.596951e+04	--	Inf	1.85s
0	1	--	534	1.594166e+04	--	Inf	1.93s
0	1	--	558	1.589388e+04	--	Inf	2.02s
0	1	--	581	1.578831e+04	--	Inf	2.12s
0	1	--	629	1.574936e+04	--	Inf	2.21s
0	1	--	623	1.574936e+04	--	Inf	2.21s

	Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
H	0	1	--	487	1.572927e+04	--	Inf	2.30s
H	0	1	--	487	1.572927e+04	7.382500e+03	53.07%	2.30s
H	0	1	--	487	1.572927e+04	7.385000e+03	53.05%	2.30s
	0	1	--	558	1.570550e+04	7.385000e+03	52.98%	2.36s
	0	1	--	558	1.570550e+04	7.385000e+03	52.98%	2.37s
	0	1	--	622	1.568897e+04	7.385000e+03	52.93%	2.45s
	0	1	--	408	1.566897e+04	7.385000e+03	52.87%	2.54s
	0	1	--	408	1.566897e+04	7.385000e+03	52.87%	2.55s
	0	1	--	508	1.564693e+04	7.385000e+03	52.80%	2.62s
	0	1	--	565	1.564002e+04	7.385000e+03	52.78%	2.70s
	0	1	--	565	1.561493e+04	7.385000e+03	52.71%	2.81s
	0	1	--	578	1.559980e+04	7.385000e+03	52.66%	2.92s
	0	1	--	571	1.558514e+04	7.385000e+03	52.62%	3.01s
H	0	1	--	571	1.558514e+04	7.387500e+03	52.60%	3.01s
	0	1	--	579	1.557589e+04	7.387500e+03	52.57%	3.08s

	Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
H	0	1	--	584	1.556642e+04	7.387500e+03	52.54%	3.17s
H	0	1	--	584	1.556642e+04	7.390000e+03	52.53%	3.17s
	0	1	--	629	1.551764e+04	7.390000e+03	52.38%	3.27s
	0	1	--	629	1.551764e+04	7.390000e+03	52.38%	3.27s
	0	1	--	612	1.551070e+04	7.390000e+03	52.36%	3.36s
	0	1	--	636	1.549185e+04	7.390000e+03	52.30%	3.45s
	0	1	--	602	1.548590e+04	7.390000e+03	52.28%	3.54s
	0	1	--	297	1.547392e+04	7.390000e+03	52.24%	3.62s
	0	1	--	625	1.546810e+04	7.390000e+03	52.22%	3.69s
	0	1	--	625	1.546810e+04	7.390000e+03	52.22%	3.70s
	0	1	--	607	1.544165e+04	7.390000e+03	52.14%	3.83s
	0	1	--	556	1.543812e+04	7.390000e+03	52.13%	3.93s
	0	1	--	597	1.542901e+04	7.390000e+03	52.10%	4.04s
	0	1	--	576	1.542851e+04	7.390000e+03	52.10%	4.11s
	0	1	--	525	1.542460e+04	7.390000e+03	52.09%	4.20s

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
0	1	--	560	1.542094e+04	7.390000e+03	52.08%	4.29s
0	1	--	586	1.542012e+04	7.390000e+03	52.08%	4.39s
0	1	--	609	1.541710e+04	7.390000e+03	52.07%	4.48s
0	1	--	602	1.541710e+04	7.390000e+03	52.07%	4.54s
0	1	--	638	1.541688e+04	7.390000e+03	52.07%	4.62s
0	1	--	669	1.541595e+04	7.390000e+03	52.06%	4.73s
0	1	--	256	1.541365e+04	7.390000e+03	52.06%	4.81s
0	1	--	575	1.540770e+04	7.390000e+03	52.04%	4.89s
0	1	--	251	1.540715e+04	7.390000e+03	52.04%	4.94s
0	1	--	260	1.540715e+04	7.390000e+03	52.04%	4.96s
0	1	--	267	1.540715e+04	7.390000e+03	52.04%	4.99s
0	1	--	525	1.540626e+04	7.390000e+03	52.03%	5.04s
0	1	--	568	1.540526e+04	7.390000e+03	52.03%	5.10s
0	1	--	515	1.540281e+04	7.390000e+03	52.02%	5.16s
0	1	--	584	1.540198e+04	7.390000e+03	52.02%	5.22s

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
0	1	--	606	1.540058e+04	7.390000e+03	52.01%	5.30s
0	1	--	362	1.539863e+04	7.390000e+03	52.01%	5.37s
0	1	--	341	1.534633e+04	7.390000e+03	51.85%	5.48s
0	1	--	319	1.534458e+04	7.390000e+03	51.84%	5.52s
0	1	--	531	1.533563e+04	7.390000e+03	51.81%	5.60s
0	1	--	317	1.533155e+04	7.390000e+03	51.80%	5.66s
0	1	--	547	1.533024e+04	7.390000e+03	51.79%	5.71s
0	1	--	477	1.532884e+04	7.390000e+03	51.79%	5.77s
0	1	--	591	1.532488e+04	7.390000e+03	51.78%	5.84s
0	1	--	486	1.532405e+04	7.390000e+03	51.78%	5.90s
0	1	--	488	1.532401e+04	7.390000e+03	51.78%	5.95s
0	1	--	358	1.532300e+04	7.390000e+03	51.77%	6.00s
0	1	--	601	1.531283e+04	7.390000e+03	51.74%	6.08s
0	1	--	626	1.531121e+04	7.390000e+03	51.73%	6.17s
0	1	--	630	1.531032e+04	7.390000e+03	51.73%	6.24s

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
0	1	--	649	1.530644e+04	7.390000e+03	51.72%	6.33s
0	1	--	330	1.530613e+04	7.390000e+03	51.72%	6.40s
0	1	--	330	1.530613e+04	7.390000e+03	51.72%	6.40s
0	1	--	614	1.530545e+04	7.390000e+03	51.72%	6.48s
0	1	--	626	1.530483e+04	7.390000e+03	51.71%	6.55s
0	1	--	600	1.530478e+04	7.390000e+03	51.71%	6.61s
0	1	--	569	1.530472e+04	7.390000e+03	51.71%	6.68s
0	1	--	598	1.530443e+04	7.390000e+03	51.71%	6.76s
0	1	--	632	1.530409e+04	7.390000e+03	51.71%	6.87s
0	1	--	632	1.530409e+04	7.390000e+03	51.71%	6.94s
0	1	--	343	1.530400e+04	7.390000e+03	51.71%	7.01s
0	1	--	344	1.529908e+04	7.390000e+03	51.70%	7.09s
0	1	--	369	1.529553e+04	7.390000e+03	51.69%	7.18s

0	1	--	674	1.529381e+04	7.390000e+03	51.68%	7.29s
0	1	--	676	1.529148e+04	7.390000e+03	51.67%	7.41s

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time	
0	1	--	654	1.529103e+04	7.390000e+03	51.67%	7.50s	
0	1	--	655	1.529103e+04	7.390000e+03	51.67%	7.57s	
0	1	--	654	1.529050e+04	7.390000e+03	51.67%	7.68s	
0	1	--	709	1.528977e+04	7.390000e+03	51.67%	7.78s	
0	1	--	675	1.528952e+04	7.390000e+03	51.67%	7.89s	
0	1	--	682	1.528937e+04	7.390000e+03	51.67%	7.99s	
0	1	--	733	1.528916e+04	7.390000e+03	51.67%	8.10s	
0	1	--	704	1.528910e+04	7.390000e+03	51.66%	8.21s	
0	1	--	704	1.528910e+04	7.390000e+03	51.66%	8.28s	
0	1	--	540	1.528907e+04	7.390000e+03	51.66%	8.39s	
H	0	0	--	540	1.528118e+04	9.182500e+03	39.91%	9.46s
H	0	0	--	540	1.528118e+04	9.185000e+03	39.89%	9.46s
H	0	0	--	540	1.528118e+04	9.192500e+03	39.84%	9.46s
H	0	0	--	540	1.528118e+04	9.200000e+03	39.80%	9.46s
	1	2	27687	540	1.528118e+04	9.200000e+03	39.80%	9.46s

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time	
2	2	14144	591	1.528117e+04	9.200000e+03	39.80%	9.62s	
3	4	9653	177	1.528117e+04	9.200000e+03	39.80%	9.62s	
4	2	7417	211	1.527813e+04	9.200000e+03	39.78%	9.83s	
5	4	5946	173	1.527813e+04	9.200000e+03	39.78%	9.83s	
6	6	5041	626	1.527813e+04	9.200000e+03	39.78%	9.83s	
7	8	4334	185	1.527813e+04	9.200000e+03	39.78%	9.83s	
8	2	3811	61	1.516205e+04	9.200000e+03	39.32%	10.19s	
9	4	3390	166	1.516205e+04	9.200000e+03	39.32%	10.19s	
10	6	3116	195	1.516205e+04	9.200000e+03	39.32%	10.19s	
20	9	1625	129	1.494378e+04	9.200000e+03	38.44%	10.84s	
*	22	11	1499	71	1.494378e+04	1.195000e+04	20.03%	10.84s
30	28	1156	174	1.494378e+04	1.195000e+04	20.03%	10.84s	
40	13	895.3	86	1.481174e+04	1.195000e+04	19.32%	11.77s	
50	31	797.5	183	1.481174e+04	1.195000e+04	19.32%	11.77s	
60	47	689.4	163	1.481174e+04	1.195000e+04	19.32%	11.78s	

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
70	33	616.5	732	1.481174e+04	1.195000e+04	19.32%	12.75s
80	47	561.7	188	1.481174e+04	1.195000e+04	19.32%	12.75s
90	62	520.2	186	1.481174e+04	1.195000e+04	19.32%	12.75s
100	44	489.7	117	1.468317e+04	1.195000e+04	18.61%	13.36s
110	61	461.0	158	1.468317e+04	1.195000e+04	18.61%	13.36s
120	74	428.9	185	1.468317e+04	1.195000e+04	18.61%	13.36s
130	56	409.5	125	1.460166e+04	1.195000e+04	18.16%	14.34s
140	68	388.2	64	1.460166e+04	1.195000e+04	18.16%	14.34s
150	83	368.3	172	1.460166e+04	1.195000e+04	18.16%	14.34s
160	65	370.0	182	1.451267e+04	1.195000e+04	17.66%	15.14s

170	79	353.6	455	1.451267e+04	1.195000e+04	17.66%	15.14s
180	93	336.3	245	1.451267e+04	1.195000e+04	17.66%	15.14s
190	73	324.6	80	1.439699e+04	1.195000e+04	17.00%	15.74s
200	91	311.4	77	1.439699e+04	1.195000e+04	17.00%	15.74s
H	265	270.1	49	1.435389e+04	1.246000e+04	13.19%	16.70s

	Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
*	300	145	250.1	37	1.434562e+04	1.246000e+04	13.14%	16.97s
*	303	147	247.8	0	1.434562e+04	1.247000e+04	13.07%	16.97s
H	332	146	233.4	0	1.434562e+04	1.271000e+04	11.40%	17.37s
	343	157	226.6	2	1.434562e+04	1.273000e+04	11.26%	17.37s
	400	139	207.0	86	1.428222e+04	1.273000e+04	10.87%	17.89s
	500	116	181.0	193	1.400213e+04	1.273000e+04	9.085%	18.62s
	600	126	161.3	109	1.397917e+04	1.273000e+04	8.936%	19.15s
	700	121	149.2	149	1.390216e+04	1.273000e+04	8.432%	19.98s
	800	98	143.3	77	1.376603e+04	1.273000e+04	7.526%	22.46s
	900	107	138.8	73	1.369804e+04	1.273000e+04	7.067%	24.73s
	1000	119	134.3	166	1.367816e+04	1.273000e+04	6.932%	26.05s
	1100	121	125.5	258	1.367384e+04	1.273000e+04	6.903%	26.42s
	1200	103	119.1	135	1.358385e+04	1.273000e+04	6.286%	27.54s
	1300	85	114.2	143	1.352272e+04	1.273000e+04	5.862%	28.31s
	1400	66	116.1	566	1.348482e+04	1.273000e+04	5.598%	29.39s

Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap	Time
1500	75	114.4	126	1.341182e+04	1.273000e+04	5.084%	30.22s
1600	54	113.1	86	1.335188e+04	1.273000e+04	4.658%	31.13s
1700	37	109.6	144	1.326264e+04	1.273000e+04	4.016%	31.70s
1800	37	106.7	85	1.324526e+04	1.273000e+04	3.890%	32.19s
1900	20	104.2	291	1.321532e+04	1.273000e+04	3.672%	32.81s
2000	21	101.0	245	1.321245e+04	1.273000e+04	3.652%	33.16s
2189	0	96.0	348	1.273000e+04	1.273000e+04	0.000%	34.82s

Best solution : 12730.000000000
 Best bound : 12730.000000000
 Best gap : 0.0000%
 Solve time : 34.82
 Solve node : 2189
 MIP status : solved
 Solution status : integer optimal (relative gap limit 0.0001)

Violations : absolute relative
 bounds : 0 0
 rows : 0 0
 integrality : 0

```
[14]: if model.status == COPT.OPTIMAL:
    print(f'The Maximum capital when arrived at destination: {model.objval}')
```

```
print(f'Solving Time: {model.SolvingTime} seconds')
```

The Maximum capital when arrived at destination: 12730.0
Solving Time: 34.818594217300415 seconds

```
[15]: def output_simple_movement(model):
    if model.status != COPT.OPTIMAL:
        return

    locations = {}
    daily_funds = {}
    for var in model.getVars():
        if var.name.startswith('x(') and abs(var.x) > 1e-6:
            try:
                indices = var.name[2:-1].split(',')
                day = int(indices[0])
                location = int(indices[1])
                locations[day] = location
            except:
                continue

        if var.name.startswith('s(') and abs(var.x) > 1e-6:
            try:
                day = int(var.name[2:-1].strip())
                amount = var.x
                daily_funds[day] = amount
            except:
                continue

    if locations:
        print("\nThe Optimal Path:")
        for day in sorted(locations.keys()):
            print(f"Day {day} → Region {locations[day]}: Left Capital ↵{daily_funds[day]}")
```

```
[16]: output_simple_movement(model)
```

The Optimal Path:

Day 0 → Region 1: Left Capital 5300.0
Day 1 → Region 2: Left Capital 5300.0
Day 2 → Region 3: Left Capital 5300.0
Day 3 → Region 4: Left Capital 5300.0
Day 4 → Region 4: Left Capital 5300.0
Day 5 → Region 5: Left Capital 5300.0
Day 6 → Region 13: Left Capital 5300.0
Day 7 → Region 13: Left Capital 5300.0
Day 8 → Region 22: Left Capital 5300.0

Day 9 → Region 30: Left Capital 5300.0
Day 10 → Region 39: Left Capital 5200.0
Day 11 → Region 39: Left Capital 3460.0
Day 12 → Region 46: Left Capital 3460.0
Day 13 → Region 55: Left Capital 3460.0
Day 14 → Region 55: Left Capital 4460.0
Day 15 → Region 55: Left Capital 5460.0
Day 16 → Region 55: Left Capital 6460.0
Day 17 → Region 55: Left Capital 7460.0
Day 18 → Region 55: Left Capital 8460.0
Day 19 → Region 62: Left Capital 4730.0
Day 20 → Region 55: Left Capital 4730.0
Day 21 → Region 55: Left Capital 5730.0
Day 22 → Region 55: Left Capital 6730.0
Day 23 → Region 55: Left Capital 7730.0
Day 24 → Region 55: Left Capital 8730.0
Day 25 → Region 55: Left Capital 9730.0
Day 26 → Region 55: Left Capital 10730.0
Day 27 → Region 55: Left Capital 11730.0
Day 28 → Region 55: Left Capital 12730.0
Day 29 → Region 56: Left Capital 12730.0
Day 30 → Region 64: Left Capital 12730.0