源代码：

# -\*- coding: utf-8 -\*-  
"""  
单场景船舶航线规划和调度的混合整数非线性规划模型  
使用 Gurobi 求解，包含二次约束和非凸约束  
"""  
from gurobipy import Model, GRB, quicksum  
import numpy as np  
  
# 数据输入  
  
# 港口数量  
N = 6 # 示例中的港口数量  
  
# 集合  
P = range(1, N + 1) # 港口集合 [1,2,3,4,5,6]  
P\_plus = range(1, N + 2) # 包含最后一个虚拟港口 N+1  
# 时间窗  
K = [1, 2, 3, 4]  
  
# 定义 P\_，用于那些从港口 p 到 p+1 的距离已知的情况  
P\_ = range(1, N) # [1,2,3,4,5]  
  
# 参数  
  
# 时间窗口和处理效率  
a = {  
 (1, 1): 0, (1, 2): 10, (1, 3): 18, (1, 4): 26,  
 (2, 1): 10, (2, 2): 18, (2, 3): 26, (2, 4): 35,  
 (3, 1): 52, (3, 2): 62, (3, 3): 72, (3, 4): 84,  
 (4, 1): 72, (4, 2): 84, (4, 3): 94, (4, 4): 106,  
 (5, 1): 94, (5, 2): 106, (5, 3): 118, (5, 4): 128,  
 (6, 1): 110, (6, 2): 125, (6, 3): 140, (6, 4): 155  
}  
  
# 时间窗口的结束时间  
b = {  
 (1, 1): 6, (1, 2): 16, (1, 3): 24, (1, 4): 32,  
 (2, 1): 16, (2, 2): 24, (2, 3): 32, (2, 4): 41,  
 (3, 1): 58, (3, 2): 68, (3, 3): 78, (3, 4): 90,  
 (4, 1): 78, (4, 2): 90, (4, 3): 102, (4, 4): 114,  
 (5, 1): 102, (5, 2): 114, (5, 3): 125, (5, 4): 132,  
 (6, 1): 118, (6, 2): 134, (6, 3): 149, (6, 4): 165  
}  
  
h = {  
 (1, 1): 240, (1, 2): 210, (1, 3): 245, (1, 4): 174,  
 (2, 1): 209, (2, 2): 241, (2, 3): 206, (2, 4): 210,  
 (3, 1): 159, (3, 2): 237, (3, 3): 157, (3, 4): 235,  
 (4, 1): 224, (4, 2): 231, (4, 3): 188, (4, 4): 212,  
 (5, 1): 245, (5, 2): 183, (5, 3): 205, (5, 4): 195,  
 (6, 1): 170, (6, 2): 186, (6, 3): 239, (6, 4): 209  
}  
  
c = {  
 (1, 1): 201, (1, 2): 169, (1, 3): 212, (1, 4): 215,  
 (2, 1): 171, (2, 2): 159, (2, 3): 227, (2, 4): 234,  
 (3, 1): 231, (3, 2): 178, (3, 3): 214, (3, 4): 219,  
 (4, 1): 235, (4, 2): 217, (4, 3): 198, (4, 4): 197,  
 (5, 1): 194, (5, 2): 248, (5, 3): 239, (5, 4): 195,  
 (6, 1): 164, (6, 2): 211, (6, 3): 163, (6, 4): 174  
}  
  
# 成本  
C0 = 625 # 固定运营成本（$/h）  
C1 = 0.5 # 库存成本（$/TEU·h）  
C\_late = {1: 500, 2: 255, 3: 260, 4: 300, 5: 320, 6: 290} # 在港口 p 的迟到惩罚成本（$/h）  
F\_q = 1000 # 加油的固定成本（$）  
  
# 燃油参数  
# 低硫燃油价格（$/吨）  
P\_M = {  
 1: 705,  
 2: 721,  
 3: 715,  
 4: 736,  
 5: 667,  
 6: 705  
}  
  
# 高硫燃油价格（$/吨）  
P\_H = {  
 1: 504,  
 2: 507,  
 3: 508,  
 4: 505,  
 5: 491,  
 6: 496  
}  
  
alpha\_M = 3.081  
alpha\_H = 3.0  
F = 1.71 # 燃油消耗系数  
v\_max = 25 # 最大航速（节）  
v\_min = 10 # 最小航速（节）  
  
# 距离参数  
l\_p = {1: 193.89, 2: 448.95, 3: 186.94, 4: 221.12, 5: 28.75} # 从港口 p 到 p+1 的不绕行距离（海里）  
d\_p = {1: 12, 2: 12, 3: 12, 4: 12, 5: 12, 6: 12} # 到达ECA边界的垂直距离（海里）  
L\_p = {1: 78.29, 2: 364.47, 3: 143.09, 4: 162.53, 5: 15.66} # 绕行距离参数  
  
# 货物和库存  
W\_p1 = {1: 1027, 2: 750, 3: 30, 4: 13, 5: 670, 6: 350}  
W\_p2 = {1: 800, 2: 900, 3: 45, 4: 15, 5: 740, 6: 320}  
W\_sp = {1: 3000, 2: 2773, 3: 2923, 4: 2898, 5: 2896, 6: 2926}  
  
# 燃油容量  
L\_M = 100 # 低硫燃油最大容量（吨）  
L\_H = 100 # 高硫燃油最大容量（吨）  
S\_M = 30 # 低硫燃油最小容量（吨）  
S\_H = 30 # 高硫燃油最小容量（吨）  
B\_M1 = 40 # 到达港口 1 时的初始低硫燃油量（吨）  
B\_H1 = 45 # 到达港口 1 时的初始高硫燃油量（吨）  
  
# 大 M 值  
M\_big = 1e7  
  
# 定义一个非常小的正数 epsilon  
epsilon = 1e-6  
  
  
T\_arr = {1: 8.0, 2: 24, 3: 68, 4: 94, 5:102, 6: 110}  
T\_dep = {1: 16, 2: 28, 3: 79, 4: 94, 5:108, 6: 114}  
T\_dep\_N = 100.0 # 港口 N 的已知离开时间  
Q = 20.0 # 参数 Q  
T\_sail = {1: 4, 2: 40, 3: 15, 4: 8, 5: 2}  
  
# 创建 Gurobi 模型  
model = Model("SingleScenarioShipRouting")  
model.Params.OutputFlag = 1  
model.Params.NonConvex = 2 # 允许非凸二次约束  
model.Params.MIPGap = 0.1  
model.Params.Threads = 6  
  
# 变量定义  
# 时间窗口变量  
x = model.addVars(P, K, vtype=GRB.CONTINUOUS, lb=0, ub=1, name="x")  
# 二进制变量 z[p,k]  
z = model.addVars(P, K, vtype=GRB.BINARY, name="z")  
  
# 二阶段变量  
t\_arr = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="t\_arr")  
t\_dep = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="t\_dep")  
t\_wait = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="t\_wait")  
t\_late = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="t\_late")  
t\_sail = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, name="t\_sail")  
t\_stay = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="t\_stay")  
  
# 新增变量  
t\_arr\_mod = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, ub=372.6, name="t\_arr\_mod")  
n\_days\_arrival = model.addVars(P, vtype=GRB.INTEGER, lb=0, name="n\_days\_arrival")  
t\_port\_entry = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="t\_port\_entry")  
t\_port\_entry\_mod = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, ub=372.6, name="t\_port\_entry\_mod")  
# 新的 n\_days\_port\_entry[p,k]  
n\_days\_port\_entry = model.addVars(P, K, vtype=GRB.INTEGER, lb=0, name="n\_days\_port\_entry")  
  
v\_eca = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=v\_min, ub=v\_max, name="v\_eca")  
v\_neca = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=v\_min, ub=v\_max, name="v\_neca")  
y = model.addVars(P\_, vtype=GRB.BINARY, name="y") # 保持为二进制变量  
x\_b = model.addVars(P, vtype=GRB.BINARY, name="x\_b") # 保持为二进制变量  
  
Q\_M = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="Q\_M")  
Q\_H = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="Q\_H")  
  
q\_M\_arr = model.addVars(P\_plus, vtype=GRB.CONTINUOUS, lb=0, name="q\_M\_arr")  
q\_H\_arr = model.addVars(P\_plus, vtype=GRB.CONTINUOUS, lb=0, name="q\_H\_arr")  
q\_M\_dep = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="q\_M\_dep")  
q\_H\_dep = model.addVars(P, vtype=GRB.CONTINUOUS, lb=0, name="q\_H\_dep")  
  
# 燃油消耗变量  
R\_M = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, name="R\_M")  
R\_H = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, name="R\_H")  
  
# 航行距离变量  
l\_e = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, name="l\_e") # ECA区域内航行距离  
l\_ne = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, name="l\_ne") # 非ECA区域航行距离  
theta = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0.01, ub=(np.pi / 2) - 0.01, name="theta") # 绕行角度  
  
# 辅助变量：cos(theta) 和 tan(theta)  
cos\_theta = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, ub=1, name="cos\_theta")  
tan\_theta = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, name="tan\_theta")  
sin\_theta = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, ub=1, name="sin\_theta")  
cot\_theta = model.addVars(P\_, vtype=GRB.CONTINUOUS, lb=0, name="cot\_theta")  
  
# 固定初始到达时间  
model.addConstr(t\_arr[1] == T\_arr[1] + Q, name="InitialArrivalTime")  
  
# t\_late[p] = t\_arr[p] - T\_arr[p]  
model.addConstrs((t\_late[p] == t\_arr[p] - T\_arr[p] for p in P), name="LateTimeDefinition")  
  
# 时间窗口约束  
for p in P:  
 # 时间窗口权重之和为 1  
 model.addConstr(quicksum(x[p, k] for k in K) == 1, name=f"TimeWindowWeight\_{p}")  
 # 只能选择一个时间窗口  
 model.addConstr(quicksum(z[p, k] for k in K) == 1, name=f"TimeWindowSelection\_{p}")  
 for k in K:  
 # x[p,k] 与 z[p,k] 的关系  
 model.addConstr(x[p, k] <= z[p, k], name=f"X\_Z\_UpperBound\_{p}\_{k}")  
 model.addConstr(x[p, k] >= epsilon \* z[p, k], name=f"X\_Z\_LowerBound\_{p}\_{k}")  
 # x[p,k] 的取值范围  
 model.addConstr(x[p, k] >= 0, name=f"X\_NonNegative\_{p}\_{k}")  
 model.addConstr(x[p, k] <= 1, name=f"X\_UpperBound\_{p}\_{k}")  
 # z[p,k] 为二进制变量，已经在变量定义中指定  
  
# 二阶段约束  
for p in P:  
 if p == 1:  
 # 港口 1 的到达时间已知  
 model.addConstr(t\_arr[1] == T\_arr[1] + Q, name=f"ArrivalTime\_{1}")  
 # 等待时间为零  
 model.addConstr(t\_wait[1] == 0, name=f"WaitingTime\_{1}")  
 # 离开时间  
 model.addConstr(t\_dep[1] == t\_arr[1] + t\_stay[1], name=f"DepartureTime\_{1}")  
 # 在港停留时间  
 model.addConstr(t\_stay[1] == quicksum((W\_p1[1] + W\_p2[1]) / h[1, k] \* x[1, k] for k in K), name=f"StayTime\_{1}")  
 else:  
 # 实际到达时间计算  
 model.addConstr(t\_arr[p] == t\_dep[p - 1] + t\_sail[p - 1], name=f"ActualArrivalTime\_{p}")  
  
 # t\_arr\_mod 和 n\_days\_arrival 的关系  
 model.addConstr(t\_arr[p] == 372.6 \* n\_days\_arrival[p] + t\_arr\_mod[p], name=f"ArrivalTimeModulo\_{p}")  
 model.addConstr(t\_arr\_mod[p] >= 0, name=f"ArrivalModNonNegative\_{p}")  
 model.addConstr(t\_arr\_mod[p] <= 372.6, name=f"ArrivalModMax\_{p}")  
  
 # 等待时间  
 model.addConstr(t\_wait[p] >= 0, name=f"WaitingTimeNonNegative\_{p}")  
  
 # 进港时间计算  
 model.addConstr(t\_wait[p] == t\_port\_entry[p] - t\_arr[p], name=f"PortEntryTime\_{p}")  
  
 # 离开时间  
 model.addConstr(t\_dep[p] == t\_port\_entry[p] + t\_stay[p], name=f"DepartureTime\_{p}")  
 # 在港停留时间  
 model.addConstr(t\_stay[p] == quicksum((W\_p1[p] + W\_p2[p]) / h[p, k] \* x[p, k] for k in K), name=f"StayTime\_{p}")  
  
 # 进港时间约束  
 for k in K:  
 # 进港时间必须在选择的时间窗口内  
  
 model.addConstr(  
 t\_port\_entry[p] >= 372.6 \* n\_days\_port\_entry[p, k] + a[p, k] - M\_big \* (1 - z[p, k]),  
 name=f"PortEntryStartWindow\_{p}\_{k}"  
 )  
 model.addConstr(  
 t\_port\_entry[p] <= 372.6 \* n\_days\_port\_entry[p, k] + b[p, k] + M\_big \* (1 - z[p, k]),  
 name=f"PortEntryEndWindow\_{p}\_{k}"  
 )  
 # 进港日期不早于到达日期  
 model.addConstr(  
 n\_days\_port\_entry[p, k] >= n\_days\_arrival[p],  
 name=f"PortEntryDays\_{p}\_{k}"  
 )  
 # 进港时间不早于到达时间  
 model.addConstr(t\_port\_entry[p] >= t\_arr[p], name=f"PortEntryAfterArrival\_{p}")  
  
# 航行时间计算（对于 p in P\_）  
for p in P\_:  
 # 定义辅助变量  
 t\_sail1 = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"t\_sail1\_{p}")  
 t\_sail2 = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"t\_sail2\_{p}")  
 t\_sail3 = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"t\_sail3\_{p}")  
 # 角度限制  
 model.addConstr(theta[p] >= 0.1, name=f"ThetaMin\_{p}")  
 model.addConstr(theta[p] <= (np.pi / 2) - 0.1, name=f"ThetaMax\_{p}")  
  
 # 定义theta的取值点和对应的cos(theta)、tan(theta)值，用于PWL近似  
 theta\_breakpoints = np.linspace(0.1, (np.pi / 2) - 0.1, 10)  
 cos\_theta\_values = np.cos(theta\_breakpoints)  
 tan\_theta\_values = np.tan(theta\_breakpoints)  
 sin\_theta\_values = np.sin(theta\_breakpoints)  
 cot\_theta\_values = 1 / tan\_theta\_values  
  
 # 添加PWL约束，定义cos\_theta和tan\_theta  
 model.addGenConstrPWL(theta[p], cos\_theta[p],  
 theta\_breakpoints.tolist(), cos\_theta\_values.tolist(),  
 name=f"CosTheta\_{p}")  
 model.addGenConstrPWL(theta[p], tan\_theta[p],  
 theta\_breakpoints.tolist(), tan\_theta\_values.tolist(),  
 name=f"TanTheta\_{p}")  
 model.addGenConstrPWL(theta[p], sin\_theta[p],  
 theta\_breakpoints.tolist(), sin\_theta\_values.tolist(),  
 name=f"SinTheta\_{p}")  
 model.addGenConstrPWL(theta[p], cot\_theta[p],  
 theta\_breakpoints.tolist(), cot\_theta\_values.tolist(),  
 name=f"CotTheta\_{p}")  
  
 # 航线距离计算  
 d\_p\_total = d\_p[p] + d\_p.get(p + 1, 0)  
 # l\_e[p] \* sin\_theta[p] == d\_p\_total \* y[p]  
 model.addConstr(l\_e[p] \* sin\_theta[p] == d\_p\_total \* y[p],  
 name=f"DistanceECA\_{p}")  
 # l\_ne[p] == (L\_p[p] - d\_p\_total \* cot\_theta[p]) \* y[p]  
 model.addConstr(l\_ne[p] == (L\_p[p] - d\_p\_total \* cot\_theta[p]) \* y[p],  
 name=f"DistanceNECA\_{p}")  
  
 # 航行时间与速度的关系  
 model.addConstr(t\_sail1 \* v\_eca[p] == l\_p[p] \* (1 - y[p]), name=f"SailTime1\_{p}")  
 model.addConstr(t\_sail2 \* v\_eca[p] == l\_e[p] \* y[p], name=f"SailTime2\_{p}")  
 model.addConstr(t\_sail3 \* v\_neca[p] == l\_ne[p] \* y[p], name=f"SailTime3\_{p}")  
  
 # 总航行时间  
 model.addConstr(t\_sail[p] == t\_sail1 + t\_sail2 + t\_sail3, name=f"TotalSailTime\_{p}")  
  
 # 定义速度比和幂次约束 v\_ratio就是v2e/v\_max  
 v\_ratio\_M = model.addVar(vtype=GRB.CONTINUOUS, lb=v\_min / v\_max, ub=1, name=f"v\_ratio\_M\_{p}")  
 v\_ratio\_H = model.addVar(vtype=GRB.CONTINUOUS, lb=v\_min / v\_max, ub=1, name=f"v\_ratio\_H\_{p}")  
 model.addConstr(v\_ratio\_M \* v\_max == v\_eca[p], name=f"VRatioM\_{p}")  
 model.addConstr(v\_ratio\_H \* v\_max == v\_neca[p], name=f"VRatioH\_{p}")  
  
 # 定义速度比的幂次变量, 这就是比值的alpha次方  
 v\_ratio\_M\_alpha = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"v\_ratio\_M\_alpha\_{p}")  
 v\_ratio\_H\_alpha = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"v\_ratio\_H\_alpha\_{p}")  
 model.addGenConstrPow(v\_ratio\_M, v\_ratio\_M\_alpha, alpha\_M, name=f"VRatioAlphaM\_{p}")  
 model.addGenConstrPow(v\_ratio\_H, v\_ratio\_H\_alpha, alpha\_H, name=f"VRatioAlphaH\_{p}")  
  
 # 定义燃油消耗距离  
 z\_p\_M = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"z\_p\_M\_{p}")  
 z\_p\_H = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"z\_p\_H\_{p}")  
  
 # 计算 z\_p\_M 和 z\_p\_H  
 model.addConstr(z\_p\_M == l\_p[p] \* (1 - y[p]) + l\_e[p] \* y[p], name=f"Z\_p\_M\_{p}")  
 model.addConstr(z\_p\_H == l\_ne[p] \* y[p], name=f"Z\_p\_H\_{p}")  
  
 # 定义燃油消耗辅助变量  
 s\_M = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"s\_M\_{p}")  
 s\_H = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"s\_H\_{p}")  
  
 # 构建燃油消耗约束  
 model.addConstr(s\_M == z\_p\_M \* v\_ratio\_M\_alpha, name=f"SM\_{p}")  
 model.addConstr(s\_H == z\_p\_H \* v\_ratio\_H\_alpha, name=f"SH\_{p}")  
  
 # 引入辅助变量 inverse\_v\_ratio\_M 和 inverse\_v\_ratio\_H  
 inverse\_v\_eca = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"inverse\_v\_eca\_{p}")  
 inverse\_v\_neca = model.addVar(vtype=GRB.CONTINUOUS, lb=0, name=f"inverse\_v\_neca\_{p}")  
  
 # 添加约束，定义这两个辅助变量为速度比的倒数  
 model.addConstr(inverse\_v\_eca \* v\_eca[p] == 1, name=f"InverseV\_M\_{p}")  
 model.addConstr(inverse\_v\_neca \* v\_neca[p] == 1, name=f"InverseV\_H\_{p}")  
  
 # 总燃油消耗公式  
 model.addConstr(R\_M[p] == F \* s\_M \* inverse\_v\_eca, name=f"FuelConsumptionM\_{p}")  
 model.addConstr(R\_H[p] == F \* s\_H \* inverse\_v\_eca, name=f"FuelConsumptionH\_{p}")  
  
# 燃油库存平衡  
for p in P:  
 if p == 1:  
 model.addConstr(q\_M\_arr[p] == B\_M1, name=f"FuelInventoryArrivalM\_{p}")  
 model.addConstr(q\_H\_arr[p] == B\_H1, name=f"FuelInventoryArrivalH\_{p}")  
 else:  
 model.addConstr(q\_M\_arr[p] == q\_M\_dep[p - 1], name=f"FuelInventoryArrivalM\_{p}")  
 model.addConstr(q\_H\_arr[p] == q\_H\_dep[p - 1], name=f"FuelInventoryArrivalH\_{p}")  
 # 燃油消耗在 P\_ 中定义，需区分 p 是否在 P\_  
 if p in P\_:  
 model.addConstr(q\_M\_dep[p] == q\_M\_arr[p] + x\_b[p] \* Q\_M[p] - R\_M[p], name=f"FuelInventoryDepartureM\_{p}")  
 model.addConstr(q\_H\_dep[p] == q\_H\_arr[p] + x\_b[p] \* Q\_H[p] - R\_H[p], name=f"FuelInventoryDepartureH\_{p}")  
 else:  
 # 对于 p = N，需要调整燃油消耗  
 model.addConstr(q\_M\_dep[p] == q\_M\_arr[p] + x\_b[p] \* Q\_M[p], name=f"FuelInventoryDepartureM\_{p}")  
 model.addConstr(q\_H\_dep[p] == q\_H\_arr[p] + x\_b[p] \* Q\_H[p], name=f"FuelInventoryDepartureH\_{p}")  
 # 燃油容量限制  
 model.addConstr(q\_M\_dep[p] >= S\_M, name=f"FuelCapacityM\_Min\_{p}")  
 model.addConstr(q\_M\_dep[p] <= L\_M, name=f"FuelCapacityM\_Max\_{p}")  
 model.addConstr(q\_H\_dep[p] >= S\_H, name=f"FuelCapacityH\_Min\_{p}")  
 model.addConstr(q\_H\_dep[p] <= L\_H, name=f"FuelCapacityH\_Max\_{p}")  
  
# 最后一个虚拟港口的燃油库存限制  
model.addConstr(q\_M\_arr[N + 1] >= S\_M, name=f"FinalFuelInventoryM")  
model.addConstr(q\_H\_arr[N + 1] >= S\_H, name=f"FinalFuelInventoryH")  
  
# 增加约束 t\_dep[N] <= Q + T\_dep[N]  
model.addConstr(t\_dep[6] <= Q + T\_dep[6], name="FinalDepartureTimeConstraint")  
  
# 目标函数  
Objective = (  
 quicksum(C0 \* t\_sail[p] for p in P\_) +  
 quicksum(F\_q \* x\_b[p] for p in P) +  
 quicksum(P\_M[p] \* Q\_M[p] + P\_H[p] \* Q\_H[p] for p in P) +  
 quicksum(C1 \* W\_sp[p] \* (t\_stay[p] + t\_wait[p] + (t\_sail[p - 1] if p != 1 else 0)) for p in P) +  
 quicksum(C\_late[p] \* t\_late[p] for p in P)  
)  
  
model.setObjective(Objective, GRB.MINIMIZE)  
  
# 求解模型  
model.optimize()  
  
# 输出结果  
if model.status == GRB.OPTIMAL:  
 print("\n模型求解成功，最优目标值为：", model.ObjVal)  
 # 初始化延误列表 E\_p  
 E\_p = {}  
 # 初始化恢复时间列表 r\_p  
 r\_p = {}  
 for p in P:  
 print(f"\n港口 {p}:")  
 print(f" 到达时间 t\_arr: {t\_arr[p].X:.2f}")  
 print(f" 离开时间 t\_dep: {t\_dep[p].X:.2f}")  
 print(f" 等待时间 t\_wait: {t\_wait[p].X:.2f}")  
 print(f" 迟到时间 t\_late: {t\_late[p].X:.2f}")  
 print(f" 停留时间 t\_stay: {t\_stay[p].X:.2f}")  
 # 时间窗口选择  
 selected\_window = max(z[p, k].X for k in K)  
 for k in K:  
 if z[p, k].X == selected\_window:  
 print(f" 选择的时间窗口: {k}")  
 break  
 # 加油策略  
 refuel\_strategy = '加油' if x\_b[p].X > 0.5 else '不加油'  
 print(f" 加油策略: {refuel\_strategy}")  
 if x\_b[p].X > 0.5:  
 print(f" 补充的低硫燃油量 Q\_M: {Q\_M[p].X:.2f}")  
 print(f" 补充的高硫燃油量 Q\_H: {Q\_H[p].X:.2f}")  
 print(f" 到达时的低硫燃油量 q\_M\_arr: {q\_M\_arr[p].X:.2f}")  
 print(f" 到达时的高硫燃油量 q\_H\_arr: {q\_H\_arr[p].X:.2f}")  
 print(f" 离开时的低硫燃油量 q\_M\_dep: {q\_M\_dep[p].X:.2f}")  
 print(f" 离开时的高硫燃油量 q\_H\_dep: {q\_H\_dep[p].X:.2f}")  
 if p in P\_:  
 print(f" 航行时间 t\_sail: {t\_sail[p].X:.2f}")  
 print(f" 从港口 {p} 到港口 {p+1} 的 ECA 航速 v\_eca: {v\_eca[p].X:.2f}")  
 print(f" 从港口 {p} 到港口 {p+1} 的非 ECA 航速 v\_neca: {v\_neca[p].X:.2f}")  
 detour\_strategy = '绕行' if y[p].X > 0.5 else '直接航线'  
 print(f" 绕行策略: {detour\_strategy}")  
 print(f" 低硫燃油消耗 R\_M: {R\_M[p].X:.2f}")  
 print(f" 高硫燃油消耗 R\_H: {R\_H[p].X:.2f}")  
 # 计算延误时间 E\_p  
 if p == 1:  
 E\_p[p] = Q + t\_sail[p].X - T\_sail[p]  
 else:  
 E\_p[p] = t\_sail[p].X - T\_sail[p]  
 else:  
 print(f" 没有后续航段。")  
 E\_p[p] = 0 # 最后一个港口没有航行时间  
 # 计算每一段的恢复时间 r\_p  
 total\_recovery = 0  
 for p in P\_:  
 if p == 1:  
 r\_p[p] = 0 # 第一段没有恢复  
 else:  
 r\_p[p] = (E\_p[p - 1] - E\_p[p])  
 total\_recovery += r\_p[p]  
 # 输出恢复时间  
 print("\n每一段的船期恢复时间（r\_p）：")  
 for p in P\_:  
 print(f" 第 {p} 段（从港口 {p} 到港口 {p+1}）：恢复时间 r\_{p} = {r\_p[p]:.2f} 小时")  
 print(f"\n总的恢复时间：{total\_recovery:.2f} 小时，初始延误 Q = {Q:.2f} 小时")  
 if abs(total\_recovery - Q) > 0 :  
 print("总恢复时间大于初始延误时间，船舶已完全恢复延误。")  
 else:  
 print("总恢复时间小于初始延误时间，船舶未能完全恢复延误。")  
else:  
 print("模型未找到最优解。")

运行结果：  
F:\soft\_hsf\anaconda\envs\keyan\python.exe F:\hsf\科研训练\ship\_route\py\科研训练.py

Set parameter NonConvex to value 2

Set parameter MIPGap to value 0.1

Set parameter Threads to value 6

Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 11.0 (22631.2))

CPU model: 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz, instruction set [SSE2|AVX|AVX2|AVX512]

Thread count: 6 physical cores, 12 logical processors, using up to 6 threads

Optimize a model with 288 rows, 300 columns and 584 nonzeros

Model fingerprint: 0x8b31b21f

Model has 77 quadratic constraints

Model has 30 general constraints

Variable types: 235 continuous, 65 integer (35 binary)

Coefficient statistics:

Matrix range [1e-06, 1e+07]

QMatrix range [1e+00, 2e+01]

QLMatrix range [1e+00, 4e+02]

Objective range [3e+02, 2e+03]

Bounds range [1e-02, 4e+02]

RHS range [1e-01, 1e+07]

QRHS range [1e+00, 4e+02]

PWLCon x range [3e-01, 1e+00]

PWLCon y range [2e-01, 4e+00]

Warning: Model contains large matrix coefficient range

Consider reformulating model or setting NumericFocus parameter

to avoid numerical issues.

Presolve added 0 rows and 196 columns

Presolve removed 158 rows and 0 columns

Presolve time: 0.01s

Presolved: 398 rows, 548 columns, 1751 nonzeros

Presolved model has 54 SOS constraint(s)

Presolved model has 49 bilinear constraint(s)

Solving non-convex MIQCP

Variable types: 519 continuous, 29 integer (29 binary)

Root relaxation: objective 1.533606e+05, 136 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 153360.571 0 40 - 153360.571 - - 0s

0 0 153720.742 0 35 - 153720.742 - - 0s

0 0 153826.939 0 36 - 153826.939 - - 0s

0 0 153826.939 0 38 - 153826.939 - - 0s

0 0 153826.939 0 36 - 153826.939 - - 0s

0 0 155969.708 0 48 - 155969.708 - - 0s

0 0 155969.708 0 48 - 155969.708 - - 0s

0 0 155972.794 0 49 - 155972.794 - - 0s

0 0 155977.962 0 49 - 155977.962 - - 0s

0 0 155978.485 0 50 - 155978.485 - - 0s

0 0 155979.049 0 50 - 155979.049 - - 0s

0 0 155979.939 0 50 - 155979.939 - - 0s

0 0 155979.939 0 50 - 155979.939 - - 0s

0 0 155979.939 0 42 - 155979.939 - - 0s

0 0 155979.939 0 42 - 155979.939 - - 0s

0 0 155980.337 0 45 - 155980.337 - - 0s

0 0 155982.467 0 45 - 155982.467 - - 0s

0 0 155989.396 0 44 - 155989.396 - - 0s

0 0 155989.396 0 44 - 155989.396 - - 0s

0 0 155989.396 0 44 - 155989.396 - - 0s

0 2 155989.396 0 44 - 155989.396 - - 0s

H 902 707 178855.01260 157884.234 11.7% 6.1 0s

H 3676 1757 169215.26861 159220.804 5.91% 9.2 0s

Cutting planes:

Gomory: 40

Implied bound: 1

MIR: 26

Flow cover: 16

RLT: 42

Relax-and-lift: 1

Explored 3744 nodes (35190 simplex iterations) in 0.86 seconds (0.65 work units)

Thread count was 6 (of 12 available processors)

Solution count 2: 169215 178855

Optimal solution found (tolerance 1.00e-01)

Warning: max constraint violation (6.3835e-04) exceeds tolerance

Warning: max general constraint violation (6.3835e-04) exceeds tolerance

Best objective 1.692152686142e+05, best bound 1.592208044930e+05, gap 5.9064%

模型求解成功，最优目标值为： 169215.2686141931

港口 1:

到达时间 t\_arr: 28.00

离开时间 t\_dep: 35.46

等待时间 t\_wait: 0.00

迟到时间 t\_late: 20.00

停留时间 t\_stay: 7.46

选择的时间窗口: 3

加油策略: 不加油

到达时的低硫燃油量 q\_M\_arr: 40.00

到达时的高硫燃油量 q\_H\_arr: 45.00

离开时的低硫燃油量 q\_M\_dep: 37.52

离开时的高硫燃油量 q\_H\_dep: 44.27

航行时间 t\_sail: 4.35

从港口 1 到港口 2 的 ECA 航速 v\_eca: 19.58

从港口 1 到港口 2 的非 ECA 航速 v\_neca: 17.89

绕行策略: 绕行

低硫燃油消耗 R\_M: 2.48

高硫燃油消耗 R\_H: 0.73

港口 2:

到达时间 t\_arr: 39.81

离开时间 t\_dep: 48.44

等待时间 t\_wait: 0.77

迟到时间 t\_late: 15.81

停留时间 t\_stay: 7.86

选择的时间窗口: 4

加油策略: 加油

补充的低硫燃油量 Q\_M: 2.10

补充的高硫燃油量 Q\_H: 3.52

到达时的低硫燃油量 q\_M\_arr: 37.52

到达时的高硫燃油量 q\_H\_arr: 44.27

离开时的低硫燃油量 q\_M\_dep: 38.20

离开时的高硫燃油量 q\_H\_dep: 35.33

航行时间 t\_sail: 19.56

从港口 2 到港口 3 的 ECA 航速 v\_eca: 22.96

从港口 2 到港口 3 的非 ECA 航速 v\_neca: 19.38

绕行策略: 绕行

低硫燃油消耗 R\_M: 1.42

高硫燃油消耗 R\_H: 12.45

港口 3:

到达时间 t\_arr: 68.00

离开时间 t\_dep: 86.06

等待时间 t\_wait: 17.74

迟到时间 t\_late: 0.00

停留时间 t\_stay: 0.32

选择的时间窗口: 4

加油策略: 不加油

到达时的低硫燃油量 q\_M\_arr: 38.20

到达时的高硫燃油量 q\_H\_arr: 35.33

离开时的低硫燃油量 q\_M\_dep: 34.72

离开时的高硫燃油量 q\_H\_dep: 33.91

航行时间 t\_sail: 7.94

从港口 3 到港口 4 的 ECA 航速 v\_eca: 19.48

从港口 3 到港口 4 的非 ECA 航速 v\_neca: 16.33

绕行策略: 绕行

低硫燃油消耗 R\_M: 3.48

高硫燃油消耗 R\_H: 1.42

港口 4:

到达时间 t\_arr: 94.00

离开时间 t\_dep: 94.15

等待时间 t\_wait: 0.00

迟到时间 t\_late: 0.00

停留时间 t\_stay: 0.15

选择的时间窗口: 3

加油策略: 不加油

到达时的低硫燃油量 q\_M\_arr: 34.72

到达时的高硫燃油量 q\_H\_arr: 33.91

离开时的低硫燃油量 q\_M\_dep: 30.48

离开时的高硫燃油量 q\_H\_dep: 30.00

航行时间 t\_sail: 7.85

从港口 4 到港口 5 的 ECA 航速 v\_eca: 20.47

从港口 4 到港口 5 的非 ECA 航速 v\_neca: 21.76

绕行策略: 绕行

低硫燃油消耗 R\_M: 4.24

高硫燃油消耗 R\_H: 3.91

港口 5:

到达时间 t\_arr: 102.00

离开时间 t\_dep: 107.76

等待时间 t\_wait: 0.00

迟到时间 t\_late: 0.00

停留时间 t\_stay: 5.76

选择的时间窗口: 1

加油策略: 不加油

到达时的低硫燃油量 q\_M\_arr: 30.48

到达时的高硫燃油量 q\_H\_arr: 30.00

离开时的低硫燃油量 q\_M\_dep: 30.00

离开时的高硫燃油量 q\_H\_dep: 30.00

航行时间 t\_sail: 2.26

从港口 5 到港口 6 的 ECA 航速 v\_eca: 12.71

从港口 5 到港口 6 的非 ECA 航速 v\_neca: 24.21

绕行策略: 绕行

低硫燃油消耗 R\_M: 0.48

高硫燃油消耗 R\_H: 0.00

港口 6:

到达时间 t\_arr: 110.01

离开时间 t\_dep: 113.95

等待时间 t\_wait: 0.00

迟到时间 t\_late: 0.01

停留时间 t\_stay: 3.94

选择的时间窗口: 1

加油策略: 不加油

到达时的低硫燃油量 q\_M\_arr: 30.00

到达时的高硫燃油量 q\_H\_arr: 30.00

离开时的低硫燃油量 q\_M\_dep: 30.00

离开时的高硫燃油量 q\_H\_dep: 30.00

没有后续航段。

每一段的船期恢复时间（r\_p）：

第 1 段（从港口 1 到港口 2）：恢复时间 r\_1 = 0.00 小时

第 2 段（从港口 2 到港口 3）：恢复时间 r\_2 = 40.79 小时

第 3 段（从港口 3 到港口 4）：恢复时间 r\_3 = -13.38 小时

第 4 段（从港口 4 到港口 5）：恢复时间 r\_4 = -6.91 小时

第 5 段（从港口 5 到港口 6）：恢复时间 r\_5 = -0.40 小时

总的恢复时间：20.10 小时，初始延误 Q = 20.00 小时

总恢复时间大于初始延误时间，船舶已完全恢复延误。

进程已结束，退出代码为 0