Cplex 社区版限制1000变量和1000约束条件。

获得Cplex22.1.1.0学术版，无限制条件。

测试平台

Processor Intel(R) Xeon(R) w9-3495X 1.90 GHz

Installed RAM 256 GB (255 GB usable)

System type 64-bit operating system, x64-based processor

Edition Windows 11 Pro for Workstations

Version 22H2

Installed on ‎10/‎4/‎2024

OS build 22621.4317

Experience Windows Feature Experience Pack 1000.22700.1041.0

软件平台

|  |  |
| --- | --- |
| 编译器 | VS code |
| Python 版本 | 3.8.x-3.10.x |
| OS | Windows |
| CPU 核心数 | 56核心 112线程 |
| 测试时间 | 22 Oct 2024 |

代码

from docplex.mp.model import Model

import random

import matplotlib.pyplot as plt

# 设置一个随机种子

random.seed(1234)

# 创建模型，在这里我们对一天24小时进行动态优化，每30分钟采样一次

sampling\_interval = 1 # 每30分钟采样一次

minutes\_per\_day = 24 \* 6000 # 一天共有 1440 分钟

time = minutes\_per\_day // sampling\_interval # 采样点数量

# 生成随机的用户负荷和用户功率

user\_loads = [round(random.uniform(0, 1), 2) for \_ in range(time)]

user\_powers = [round(random.uniform(0, 1), 2) for \_ in range(time)]

# 生成随机电价，每2小时变动一次，基于 0.5 生成 ±0.2 的波动

elec\_price = []

for i in range(0, time, 12): # 每2小时（12个采样点）生成一个电价

price = round(0.5 + random.uniform(-0.2, 0.2), 2)

elec\_price.extend([price] \* 12)

# 设置储能充放电效率和单次充放限制

charge\_eff = 0.91

discharge\_eff = 0.95

nominal\_power = 0.8

# 创建模型

model = Model(name='Electricity')

model.parameters.threads.set(2)

# 创建变量

x = model.continuous\_var\_list(time, lb=0, ub=nominal\_power, name='x')

y = model.continuous\_var\_list(time, lb=-nominal\_power, ub=0, name='y')

soc = model.continuous\_var\_list(time, lb=0, ub=nominal\_power, name='soc')

# 无储能下的花费

cost\_base = sum(((user\_loads[i] - user\_powers[i]) \* elec\_price[i]) for i in range(time))

# 总花费

total\_cost = model.sum(((user\_loads[i] - user\_powers[i] + x[i] + y[i]) \* elec\_price[i]) for i in range(time))

# 设置目标函数

model.minimize(total\_cost)

# 添加约束条件

min\_soc = 0.2 \* nominal\_power # 最小 SOC 限制

model.add\_constraint(soc[0] == 0.5 \* nominal\_power) # 初始 SOC 设置为 0.5

for i in range(time - 1):

# SOC 状态更新约束

model.add\_constraint(soc[i + 1] == soc[i] + x[i] \* charge\_eff + y[i] / discharge\_eff)

# 最小 SOC 约束

model.add\_constraint(soc[i] >= min\_soc)

# 放电后的冷却时间约束（通过引入额外的二进制变量来表示冷却状态）

for i in range(time - 2):

# 创建二进制变量来表示冷却状态

cooling\_20 = model.binary\_var(name=f'cooling\_20\_{i}')

cooling\_10 = model.binary\_var(name=f'cooling\_10\_{i}')

# 冷却 20 分钟约束，当放电量超过 50% 时触发冷却

model.add\_indicator(cooling\_20, y[i] <= -0.5 \* nominal\_power, 1)

model.add\_constraint(x[i + 1] <= nominal\_power \* (1 - cooling\_20)) # 冷却期间不可充电

# 冷却 10 分钟约束，当放电量在 30% - 50% 之间时触发冷却

model.add\_indicator(cooling\_10, y[i] <= -0.3 \* nominal\_power, 1)

model.add\_constraint(x[i + 1] <= nominal\_power \* (1 - cooling\_10)) # 冷却期间不可充电

solution = model.solve()

# 检查是否找到最优解

if solution:

print('The optimized solution is:')

print("charging(+)/discharging(-) power for each time step:")

charging\_discharge\_values = []

soc\_values = []

for i in range(time):

charge\_discharge = x[i].solution\_value + y[i].solution\_value

charging\_discharge\_values.append(charge\_discharge)

soc\_values.append(soc[i].solution\_value)

if i % (60 // sampling\_interval) == 0: # 每小时输出一次状态

print(f"Hour {i // (60 // sampling\_interval)}: solution: {charge\_discharge}")

# 可视化结果

hours = [i \* sampling\_interval / 60 for i in range(time)]

plt.figure(figsize=(14, 10))

# 负荷、用户功率和充放电功率可视化

plt.plot(hours, user\_loads, label='User Load', color='r')

plt.plot(hours, user\_powers, label='User Power', color='m')

plt.plot(hours, charging\_discharge\_values, label='Charging/Discharging Power', color='b')

plt.xlabel('Time (hours)')

plt.ylabel('Power (kW)')

plt.title('Load, Power, and Charging/Discharging Over Time')

plt.legend()

# SOC 状态可视化

plt.plot(hours, soc\_values, label='State of Charge (SOC)', color='g')

plt.ylabel('SOC')

plt.title('State of Charge, Load, and Power Over Time')

plt.legend()

plt.tight\_layout()

plt.show()

print(f'Total cost: {total\_cost.solution\_value}')

else:

print("No solution found.")

代码存在数学上的错误但不影响测试Cplex的性能。

结果

|  |  |
| --- | --- |
| 线程数 | 计算时间 |
| 1 | 157.5s |
| 2 | 64.6s |
| 4 | 56.6s |
| 8 | 58.0s |
| 16 | 58.1s |
| 32 | 57.6s |
| 64 | 59.5s |
| 112 | 62.4s |
| 128 | 87.5s |
| 256 | 408.7s |
|  |  |
|  |  |

CPLEX只支持3.8.x~3.10.x 版本python.

CPLEX 对线程的支持是非线性的。从单线程到2,4线程，计算时间大幅减少。当启用线程数过多时，计算时间反而增加，特别是超过实际物理线程数量时。

结论：

1. 推荐使用Python 3.10版本
2. 线程数不宜过多
3. 建议选用Gurobi，其支持更新的python 版本