测试对象 Cplex (商业)，cbc, SCIP, GLPK, 在python 下的性能。

因版权问题，无法对Gurobi 进行测试，但其性能是行业周知的。

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 |
| OS | Windows |
| CPU 核心数 | 56核心 112线程 |
| 测试时间 | 29 Oct 2024 |

代码

import random

import pandas as pd

import numpy as np

import pulp

from pulp import SCIP\_CMD, PULP\_CBC\_CMD, GLPK\_CMD

import tempfile

# 设置随机种子

random.seed(1234)

# 参数设置

clear\_interval = 30

clear\_period = 24 \* 600

ctrl\_interval = 1

if clear\_period % clear\_interval != 0 or clear\_interval % ctrl\_interval != 0:

    raise ValueError("clear\_period 必须是 clear\_interval 的整数倍")

n\_clearance = clear\_period // clear\_interval

n\_ctrl = clear\_period // ctrl\_interval

# 生成随机数据

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

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

user\_loads = np.repeat(user\_loads, clear\_interval // ctrl\_interval)

user\_powers = np.repeat(user\_powers, clear\_interval // ctrl\_interval)

netload = [user\_loads[i] - user\_powers[i] for i in range(n\_ctrl)]

#print("净负荷：", netload)

elec\_price = []

for i in range(0, n\_clearance, 12):

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

    elec\_price.extend([price] \* 12)

elec\_price = np.repeat(elec\_price, clear\_interval // ctrl\_interval)

#print('电价：', elec\_price)

# 储能系统参数

charge\_eff = 0.91

discharge\_eff = 0.95

nominal\_power = 0.8

SOC\_ub = 1

SOC\_lb = 0

SOC0 = 0.5

Ckwh = 1

# 创建Pulp模型

model = pulp.LpProblem("Electricity\_Optimization", pulp.LpMinimize)

# 变量定义

x = [pulp.LpVariable(f"x\_{i}", lowBound=0, upBound=nominal\_power) for i in range(n\_ctrl)]

y = [pulp.LpVariable(f"y\_{i}", lowBound=-nominal\_power, upBound=0) for i in range(n\_ctrl)]

soc = [pulp.LpVariable(f"soc\_{i}", lowBound=SOC\_lb, upBound=SOC\_ub) for i in range(n\_ctrl)]

z = [pulp.LpVariable(f"z\_{i}", cat="Binary") for i in range(n\_ctrl)]

# 无储能下的花费

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

print("无储能下的花费：", cost\_base)

# 目标函数

total\_cost = pulp.lpSum(

    (user\_loads[i] - user\_powers[i] + x[i] + y[i]) \* elec\_price[i] for i in range(n\_ctrl)

)

model += total\_cost

# 初始 SOC 约束

model += soc[0] == SOC0

# 添加 SOC 更新和充放电限制约束

for i in range(n\_ctrl - 1):

    model += soc[i + 1] == soc[i] + x[i] \* (charge\_eff \* ctrl\_interval / 60 / Ckwh) + y[i] \* (ctrl\_interval / discharge\_eff / 60 / Ckwh)

    model += x[i + 1] <= x[i] + 0.01

    model += x[i + 1] >= x[i] - 0.01

    model += y[i + 1] <= y[i] + 0.01

    model += y[i + 1] >= y[i] - 0.01

for i in range(n\_ctrl):

    model += x[i] <= z[i] \* nominal\_power

    model += -y[i] <= (1 - z[i]) \* nominal\_power

# 创建临时文件来存储日志

with tempfile.NamedTemporaryFile(delete=False) as tmp\_file:

    log\_file\_path = tmp\_file.name

# 比较多种求解器

solvers = {

    "SCIP": SCIP\_CMD(msg=False),

    "CBC": PULP\_CBC\_CMD(msg=False), #速度暂时和SCIP差不多，求解结果一致

    #"GLPK": GLPK\_CMD(msg=False), #速度最慢，可能是应用场景不适配。

    # "Gurobi": pulp.GUROBI\_CMD(msg=True), #需要安装Gurobi软件，且需要在系统路径中添加Gurobi可执行文件路径。

    "CPLEX": pulp.CPLEX\_CMD(msg=False) #需要安装CPLEX软件，且需要在系统路径中添加CPLEX可执行文件路径。

    # "XPRESS": pulp.XPRESS\_CMD(msg=True), #需要安装XPRESS软件，且需要在系统路径中添加XPRESS可执行文件路径。

    # "Mosek": pulp.MOSEK\_CMD(msg=True), #需要安装Mosek软件，且需要在系统路径中添加Mosek可执行文件路径。

}

import time

# 初始化一个字典来保存每个求解器的求解时间和总成本

results\_summary = {}

for solver\_name, solver in solvers.items():

    print(f"\nUsing solver: {solver\_name}")

    # 开始计时

    start\_time = time.time()

    # 使用当前求解器求解模型

    model.solve(solver)

    # 结束计时并计算总时间

    end\_time = time.time()

    solve\_time = end\_time - start\_time

    # 打印求解器日志

    with open(log\_file\_path, 'r') as log\_file:

        print(log\_file.read())

    # 检查解并输出结果

    if pulp.LpStatus[model.status] == 'Optimal':

        total\_cost\_value = pulp.value(model.objective)

        print(f"{solver\_name} 求解器使花费最低化：", total\_cost\_value)

        print(f"{solver\_name} 求解时间：", solve\_time, "秒")

        # 记录结果

        results\_summary[solver\_name] = {"Total Cost": total\_cost\_value, "Solve Time": solve\_time}

    else:

        print(f"{solver\_name} 未找到可行解")

        results\_summary[solver\_name] = {"Total Cost": None, "Solve Time": solve\_time}

if results\_summary:

    best\_solver = min(results\_summary, key=results\_summary.get)

    print(f"\n最佳解由求解器 {best\_solver} 得到，总花费为：{results\_summary[best\_solver]}")

else:

    print("\n没有求解器找到最佳解")

无储能下的花费： 33.89100000000194

Using solver: SCIP

SCIP 求解器使花费最低化： -112.81447252866626

SCIP 求解时间： 67.79318141937256 秒

Using solver: CBC

CBC 求解器使花费最低化： -112.81447286804388

CBC 求解时间： 11.969951391220093 秒

Using solver: CPLEX

CPLEX 求解器使花费最低化： -112.81447252866552

CPLEX 求解时间： 21.657469987869263 秒

\*GLPK 在测试中没有找到解。

结论：

1. 开源推荐使用SCIP, CBC求解器。开源求解器的功能覆盖度，计算速度，求解准确度普遍不如商用求解器。商用推荐使用gurobi。 Cplex 的软件维护较差仅支持到python 3.10.x 版本。
2. 本项目中的模型多为非线性模型。需要将模型线性化再求解。