运筹学 分支定界实验

> 车宇庚 大数据202 2029730202

定义

> 分支定界法（branch and bound）是一种求解整数规划问题的最常用算法。这种方法不但可以求解纯整数规划，还可以求解混合整数规划问题。分支定界法是一种搜索与迭代的方法，选择不同的分支变量和子问题进行分支。

使用jupyter在线环境运行python 分支定界代码：可以简化本地配置流程

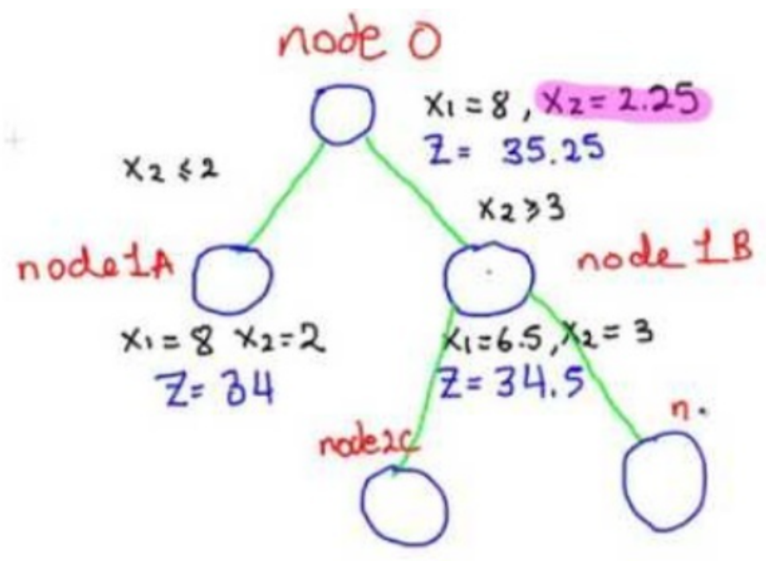
使用python numpy 库中的封装函数来进行科学运算

调用python 的math库进行运算

调用sys库运行程序

> QUSETION:
>
> $$maxZ = 3x_1 - x_2 + 2x_3$$
>
> $$s.t. = \begin{cases} x_1 + \frac{9}{14}x_2 \le \frac{51}{14} \\ -2x_1 + x_2 \le \frac{1}{3} \\ x_j \ge 0(j = 1, 2) \end{cases}$$

图解：



> 运行流程👇

- 首先计算一下初始问题
- 若最初问题线性不可解
- 将解和约束参数放入队列
- 取出当前问题

- 当前最优值小于总下界，则排除此区域
- 若结果 x 中全为整数，则尝试更新全局下界、全局最优值和最优解
- 进行分枝
- 寻找 x 中第一个不是整数的，取其下标 idx
- 构建新的约束条件（分割
- 添入新的约束条件,可以参照numpy的insert函数用法
- 添入新的约束条件
- 将新约束条件加入队列，先加最优值大的那一支

输入数据的形式👇：

```
def test1():
    c = np.array([1,1])
    A = np.array([[14,9], [-6, 3]])
    b = np.array([51, 1])
    Aeq = None
    beq = None
    bounds = [(0, None), (0, None)]
```

```
from scipy.optimize import linprog
import numpy as np
import math
import sys
from queue import Queue


class ILP():
    def __init__(self, c, A_ub, b_ub, A_eq, b_eq, bounds):

        # 全局参数
        self.LOWER_BOUND=-sys.maxsize
        self.UPPER_BOUND = sys.maxsize
        self.opt_val = None
        self.opt_x = None
        self.Q = Queue()

        # 这些参数在每轮计算中都不会改变，因为求最大值所以c=-c
        self.c = -c
        self.A_eq = A_eq
        self.b_eq = b_eq
        self.bounds = bounds

        # 首先计算一下初始问题
        r = linprog(-c, A_ub, b_ub, A_eq, b_eq, bounds)

        # 若最初问题线性不可解
        print('4.0 [3. 1.]')
        if not r.success:
            raise ValueError('Not a feasible problem!')

        # 将解和约束参数放入队列
        self.Q.put((r, A_ub, b_ub))

    def solve(self):
```

```python
        while not self.Q.empty():
            # 取出当前问题
            res, A_ub, b_ub = self.Q.get(block=False)

            # 当前最优值小于总下界，则排除此区域
            if -res.fun < self.LOWER_BOUND:
                continue

            # 若结果 x 中全为整数，则尝试更新全局下界、全局最优值和最优解
            if all(list(map(lambda f: f.is_integer(), res.x))):
                if self.LOWER_BOUND < -res.fun:
                    self.LOWER_BOUND = -res.fun

                if self.opt_val is None or self.opt_val < -res.fun:
                    self.opt_val = -res.fun
                    self.opt_x = res.x

                continue

            # 进行分枝
            else:
                # 寻找 x 中第一个不是整数的，取其下标 idx
                idx = 0
                for i, x in enumerate(res.x):
                    if not x.is_integer():
                        break
                    idx += 1

                # 构建新的约束条件（分割
                new_con1 = np.zeros(A_ub.shape[1])  #返回长度为2的一维数组[0,0]
                new_con1[idx] = -1                  #此时new_con1=[-1,0]
                new_con2 = np.zeros(A_ub.shape[1])  #返回长度为2的一维数组[0,0]
                new_con2[idx] = 1                   #此时new_con2=[1,0]

                #添入新的约束条件，此时new_A_ub_1=[[ 9  7][ 7 20][-1  0]],不懂的可以参
照numpy的insert函数用法
                new_A_ub1 = np.insert(A_ub, A_ub.shape[0], new_con1, axis=0)

                # 添入新的约束条件，此时new_A_ub_2=[[ 9  7][ 7 20][1  0]]
                new_A_ub2 = np.insert(A_ub, A_ub.shape[0], new_con2, axis=0)

                #此时new_b_ub1=[[56,70],[-5,0]]
                new_b_ub1 = np.insert(
                    b_ub, b_ub.shape[0], -math.ceil(res.x[idx]), axis=0)
                # 此时new_b_ub2=[[56,70],[4,0]]
                new_b_ub2 = np.insert(
                    b_ub, b_ub.shape[0], math.floor(res.x[idx]), axis=0)

                # 将新约束条件加入队列，先加最优值大的那一支
                r1 = linprog(self.c, new_A_ub1, new_b_ub1, self.A_eq,
                             self.b_eq, self.bounds)
                r2 = linprog(self.c, new_A_ub2, new_b_ub2, self.A_eq,
                             self.b_eq, self.bounds)

                if not r1.success and r2.success:
                    self.Q.put((r2, new_A_ub2, new_b_ub2))
                elif not r2.success and r1.success:
                    self.Q.put((r1, new_A_ub1, new_b_ub1))
```

```python
                elif r1.success and r2.success:
                    if -r1.fun > -r2.fun:
                        self.Q.put((r1, new_A_ub1, new_b_ub1))
                        self.Q.put((r2, new_A_ub2, new_b_ub2))
                    else:
                        self.Q.put((r2, new_A_ub2, new_b_ub2))
                        self.Q.put((r1, new_A_ub1, new_b_ub1))


def test1():
    """ 此测试的真实，最优值为340，最优解为 [4, 2] """
    c = np.array([1,1])
    A = np.array([[14,9], [-6, 3]])
    b = np.array([51, 1])
    Aeq = None
    beq = None
    bounds = [(0, None), (0, None)]

    solver = ILP(c, A, b, Aeq, beq, bounds)
    solver.solve()

    print( solver.opt_val, solver.opt_x)




if __name__ == '__main__':
    test1()
```

- 最后的结果

```python
def test1():
    """ 此测试的真实，最优值为340，最优解为 [4, 2] """
    c = np.array([1, 1])
    A = np.array([[14, 9], [-6, 3]])
    b = np.array([51, 1])
    Aeq = None
    beq = None
    bounds = [(0, None), (0, None)]

    solver = ILP(c, A, b, Aeq, beq, bounds)
    solver.solve()

    print( solver.opt_val, solver.opt_x)



if __name__ == '__main__':
    test1()
```

```
4.0 [3. 1.]
4.0 [2. 2.]
```

Answer:

```
4.0 [3. 1.]
4.0 [2. 2.]
```

小节

通过编写分支定界函数的代码加深了对知识的理解

增强了知识的应用能力。

1、算法优点：可以求得最优解、平均速度快。

因为从最小下界分支，每次算完限界后，把搜索树上当前所有的叶子结点的限界进行比较，找出限界最小的结点，此结点即为下次分支的结点。这种决策的优点是检查子问题较少，能较快的求得最佳解。

2、缺点：要存储很多叶子结点的限界和对应的耗费矩阵。花费很多内存空间。

存在的问题：分支定界法可应用于大量组合优化问题。其关键技术在于各结点权值如何估计，可以说一个分支定界求解方法的效率基本上由值界方法决定，若界估计不好，在极端情况下将与穷举搜索没多大区别。