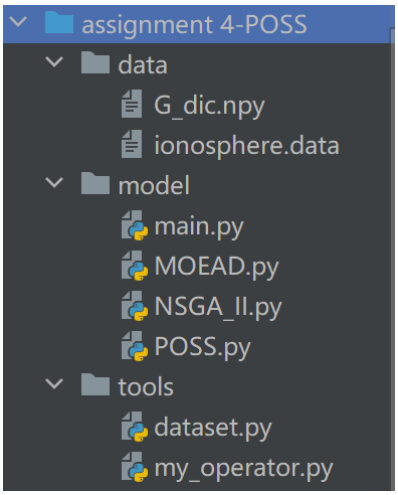
HSEA--第四次作业实验报告

201300086 史浩男

代码结构



- data中的G_dic.npy是用于最大覆盖问题的数据集，ionosphere.data是用于稀疏回归问题的数据集
- tools中的dataset.py用于读取并预处理数据集
- tools中的my_operator.py包含算法需要的所有工具函数，包括`mutation`、`crossover`、生成初始种群的函数、优化目标函数、画图函数等
- model中包含了三个算法具体是实现代码和汇总画图的main.py

子集选择问题转化

对于不同算法，问题的转化方法是相同的，因此这里先介绍对两个子集选择问题的转化

此部分代码由于比较通用，都在tools文件中

一、稀疏回归

数据集选择：论文中的ionosphere.data

- 这是个二分类数据集，有34个属性列，一个标记列
- 将34个属性列度进data，标记列读为Z（将原始标记g和b转化为1和2，方便进行计算）
- 通过矩阵乘法，计算出系数矩阵Alpha

```
data = data.T[:-1].T.astype("float32")
# 计算初始系数矩阵Alpha
DATA = data
Alpha = np.linalg.pinv(DATA) @ Z
Alpha = Alpha.astype("float32").T
```

- 由系数矩阵Alpha，可得出进行子集选择前的最优MSE=0.2987

优化目标函数

```
def f1(solution):  # 计算稀疏回归的MSE
    new_Alpha = []
    for i in range(len(solution)):
        if solution[i] == 0:
            new_Alpha.append(0)
        else:
            new_Alpha.append(Alpha[i])
    new_result = DATA @ new_Alpha
    return mean_squared_error(Z, new_result)


def f2(solution):  # 计算稀疏回归的S元素个数
    return sum(solution)
```

解的比较函数

```
def is_better_regression(p, q):
    if f1(p) == f1(q) and f2(p) == f2(q):#相同
        return 0
    elif f1(p) <= f1(q) and f2(p) <= f2(q):#p优
        return 1
    elif f1(p) >= f1(q) and f2(p) >= f2(q):#q优
        return -1
    else:#无法比较
        return 10
```

二、最大覆盖

数据集选择：对第二次作业的Gset中G1.txt进行修改，对原始的800个点的图进行了删减，将图缩小到500个点，约7800条边

- 为了加速存取数据集，我改为用字典的形式存储图（如dic[i]是一个列表，里面存储了所有与点i相连的点），并将字典存为文件G_dic.npy

优化目标函数

```
def f3(solution):  # 计算最大覆盖点集合,max
    cover_set = []
    for i in range(1, 501):
        if solution[i - 1] == 0:
            pass
        else:
            cover_set.extend(data[i])
    cover_set = set(cover_set)#去重#先列表后集合，可加速
    return len(cover_set)


def f4(solution):  # 点个数,min
    return sum(solution)
```

解的比较函数

```
def is_better_cover(p, q):
    if f3(p) == f3(q) and f4(p) == f4(q):
        return 0
    elif f3(p) >= f3(q) and f4(p) <= f4(q):
        return 1
    elif f3(p) <= f3(q) and f4(p) >= f4(q):
        return -1
    else:
        return 10
```

任务一：实现POSS算法, 用于求解子集选择问题

一、算法设计

- 随机生成指定长度的01串作为初始解（没有采用论文中的全0初始解）

  ```
  def generate_binary(n):
      seed = "01"
      sa = []
      for i in range(n):
          sa.append(random.choice(seed))
      salt = ''.join(sa)
      x = np.array(list(map(int, salt)))
      return x
  ```

- 种群大小不固定，这是POSS算法的特征
- 演化算子只有`mutation`，不包含`crossover`。采取`bit_wise_mutation`

  （这里我遇到一个低级bug，注意mutation前先深拷贝）

  ```
  def bit_wise_mutation(y, p):
      x = copy.deepcopy(y)  # 必须先copy再变
      for num, i in enumerate(x):
          seed = random.randint(0, len(x))
          if seed < int(len(x) * p):
              if x[num] == 0:
                  x[num] = 1
              else:
                  x[num] = 0
      return x
  ```

二、算法迭代过程

- 第一步我选择先对当前种群进行清洗，去除重复的解，并参照POSS中的实现，去除$|S| \geq 2k$的解；

```
for n in solution_group:
    if f1(n) in dic.keys():
        pass
    else:
        dic[f1(n)] = 1
        group2.append(n)
for i in group2:
    if f2(i) <= 2 * k:
```

- 随机在当前种群中抽取一个解用来`mutation`

```
select_index = random.randint(0, len(solution_group) - 1)

select_solution = bit_wise_mutation(solution_group[select_index], pm)
```

- 先检查种群中是否已有比变异后的解更好的，如果没有，则替换掉种群中所有比变异解差的

```
            for i in range(len(solution_group)):
                if is_better_regression(select_solution, solution_group[i])
== -1:
                    better = 1
                    break
            if better == 0:
                new_group.append(select_solution)
                for i in range(len(solution_group)):
                    if is_better_regression(select_solution,
solution_group[i]) == 10:  # 无法比较的
                        new_group.append(solution_group[i])
```

任务二：实现NSGA-II与MOEA/D算法，用于求解子集选择问题

一、NSGA-II算法设计

采用fast_non_dominated_sort

```
#当前解为p，计算：
S[p] = []  # 比当前解差的集合
n[p] = 0  # 比当前解好的个数
```

函数返回所有前沿解的index

```
if n[p] == 0:
    rank[p] = 0  # 最优
    if p not in front[0]:  # 帕累托前沿
        front[0].append(p)
```

- 初始化种群

```
solution_group = [generate_binary(len(Alpha)) for i in range(0, Group_size)]
```

二、NSGA-II算法迭代过程

- 先计算当前种群的各个前沿分布情况和distance情况

```
front1 = fast_non_dominated_sort(f1_values[:], f2_values[:], "regression")
distance = []
for i in range(0, len(front1)):
    distance.append(crowding_distance(f1_values[:], f2_values[:], front1[i]
[:]))
```

- 父辈选择：

  选取最优的几个前沿作为父辈，如果数量不够则使用全部前沿

```
parent_index = []
if len(front) >= 5:
    parent_index = front[0][:] + front[1][:] + front[2] + front[3] +
front[4]
else:
    for i in front:
        parent_index += i
parents = [solution_group[i] for i in parent_index]
```

- 产生子代：

  相邻的两个父辈，经过`mutation`和`crossover`操作后，产生两个子代

```
        for i in range(len(parents)):
            for j in range(i + 1, 2 * len(parents)):
                jj = j % len(parents)
                if f1(parents[i]) != f1(parents[jj]):
                    tem1 = bit_wise_mutation(parents[i], pm)
                    tem2 = bit_wise_mutation(parents[jj], pm)
                    off1, off2 = one_point_crossover(tem1, tem2, pc)
                    if len(offsprings) < 2 * Group_size:
                        offsprings.append(off1)
                    if len(offsprings) < 2 * Group_size:
                        offsprings.append(off2)
                    break
                else:
                    pass
```

- 幸存者选择：

  保留父辈中的帕列托前沿中所有个体作为下一轮的幸存者

```
# 优秀父辈直接继承
            old_solution = []
            for k in range(0, 1):
                for v in front1[k]:
                    old_solution.append(v)

            solution_group = [solution_group[i] for i in old_solution]#继承
```

其余的，通过distance排序，在新产生的子代中选择，直到选出`Group_size`个幸存者

三、MOEAD算法设计

产生权重向量和邻居

```
K = max(math.ceil(0.15 * population), 2)
K = min(K, 15)  # 邻居的数目


def genVector2(nObj, npop, T):
    l = []
    dist = np.zeros((npop, npop))
    for i in range(npop):
        w = np.random.rand(nObj)
        w = w / np.linalg.norm(w)
        l.append(w)
    for i in range(npop - 1):
        for j in range(i + 1, npop):
            dist[i][j] = dist[j][i] = np.linalg.norm(l[i] - l[j])
    neighbors = np.argsort(dist, axis=1)
    neighbors = neighbors[:, :T]
    return l, neighbors
```

初始化种群，初始化全局最优z

```
for i in range(population):
    var = varMin + np.random.random(nVar) * (varMax - varMin)
    cost = ZDT2(var)
    pops.append(pop(var, cost))
z = pops[0].cost
for p in range(population):
    for j in range(nObj):
        z[j] = min(pops[p].cost[j], z[j])
z = np.array(z)
determinDomination(pops)
ep = copy.deepcopy([x for x in pops if x.dominate != True])
```

更新邻域解

```
def update_neighbor(idx, y):
    Bi = sp_neighbors[idx]
    fy = y.cost
    for j in range(len(Bi)):
        w = Lambda[Bi[j]]
        maxn_y = max(w * abs(fy - z))
        maxn_x = max(w * abs(pops[Bi[j]].cost - z))
        if maxn_x >= maxn_y:
            pops[Bi[j]] = y
```

任务三：算法改进

经过众多尝试，我在两个细节的改动后成功得到了更快的收敛效果

1、改进mutation操作

原始的bit_wise_mutation是对每一位都有固定的概率p进行变异，这并不适用于子集选择问题!!

以最大覆盖问题为例，我选择的数据集有500个点，每个解都是一个01串，代表每个点是否被选择。而其中一个优化目标是$k \leq 8$，这就意味着如果采用原始的mutation方法，很难变异出满足$k \leq 8$的解，在此基础上筛选$\leq 2 * k$的解放入种群将花费极大的时间代价。

优化方案：在mutation后统计01串中1的个数，把所有的1以一定概率再变回0，控制mutation后的01串中1的个数接近8（优化目标）

优化后的mutation方式如下：

```
def bit_wise_mutation_cover(y, p):
    x = copy.deepcopy(y)  # 必须先copy再操作
    for num, i in enumerate(x):
        seed = random.random() * len(x)
        if seed < int(len(x) * p):
            if x[num] == 0:
                x[num] = 1
            else:
                x[num] = 0
    # 控制解中1数量不要太多
    now = sum(x)
    q = (now - k - 1) / now
    for num, i in enumerate(x):
        seed = random.random() * len(x)
        if seed < len(x) * q:
            if x[num] == 1:
                x[num] = 0
    return x
```

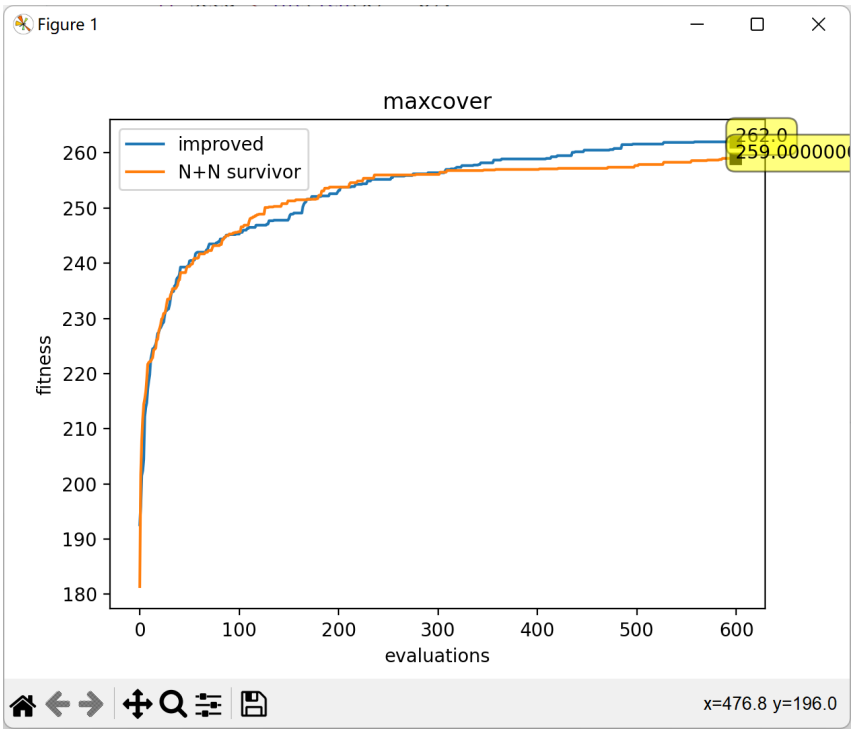
2、优化NSGA-II的幸存者选择方式

我将原始的N+N survivor selection进行了修改：

只保留父辈中的帕列托前沿中所有个体作为下一轮的幸存者，其他的幸存者都从offspring中择优选择

```
old_solution = []
for k in range(0, 1):
    for v in front1[k]:#帕列托前沿中的所有解
        old_solution.append(v)
```

迭代600轮，运行10次取平均，可以看出性能得到了一些明显的提升



汇报结果

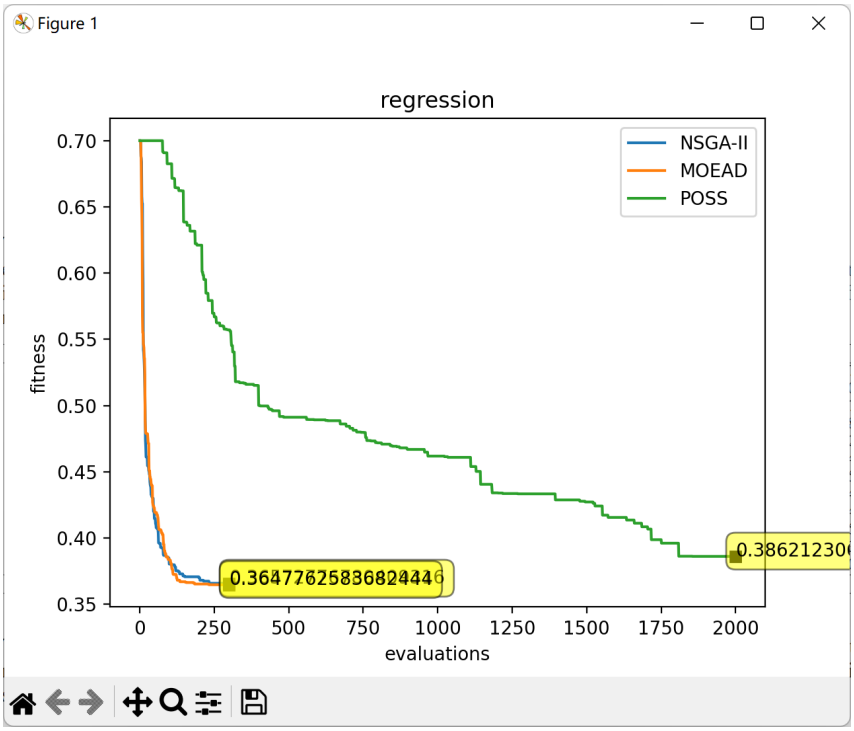
3个算法，2个问题，一共6个任务

所有任务均采用k=8，运行10次，取均值后画出evaluations--fitness曲线

其中NSGA-II的图像采用的是优化后的，所以对比图中只有三条曲线

1、稀疏回归

Dataset:ionosphere.data	POSS	NSGA-II	MOEAD
time	425s	127s	121s
evaluation	2000	300	300
fitness	0.38	0.364	0.363



2、最大覆盖

Dataset:500点，7800边的无向图	POSS	NSGA-II	MOEAD
time	250s	278s	285s
evaluation	5000	800	800
fitness	264.3	264.8	251.1