作业

Homework4

庄镇华 502022370071

A HSEA Homework Assignment

2023 年 1 月 30 日

实验环境

本次作业要求大家使用多目标演化算法求解一类带约束的子模优化问题: 子集选择问题。具体包含两种问题类型，分别为稀疏回归和最大覆盖问题。

关于数据集：1) 稀疏回归可以选一些 POSS 论文里的数据集, 可以根据你的计算资源选择, 报告中标明即可. 推荐是选择规模大一些的数据集, 因为小规模的话问题太简单, 不同算法之间对比不明显.

2) 最大覆盖的数据集可以用自己生成的图, 例如第二次作业的 nx 库来生成; 也可以做一些传统最大覆盖问题的数据集, 例如https://snap.stanford.edu/data/.

稀疏回归问题：给定所有观察变量 $V = \{v_1, \cdots, v_n\}$，一个预测变量 Z，将子集 $S \subseteq V$ 的均方误差定义为

$$MSE_{Z,S} = \min_{\alpha \in \mathbb{R}^{|S|}} \mathbb{E}\left[\left(Z - \sum_{i \in S} \alpha_i v_i\right)^2\right],$$

稀疏回归问题的目的是找出一组至多 k 个变量，使得均方误差最小化

$$\min_{S \subseteq V} MSE_{Z,S} \quad \text{s.t. } |S| \leq k.$$

最大覆盖问题：给定一组元素的集合 U，一个元素为 U 的子集的集合 $V = \{v_1, \cdots, v_n\}$ 和一个正整数 k，从 V 中找出最多 k 个子集使得涵盖的元素数最多，即

$$\arg\max_{S \subseteq V} f(S) = \left|\cup_{v_i \in S} v_i\right| \quad \text{s.t. } |S| \leq k.$$

1 POSS 算法

任务一 (40pts)

实现 POSS 算法, 用于求解子集选择问题

参考论文：Subset Selection by Pareto Optimization

- 你需要实现 POSS 算法：定义解与种群，设计演化算子
- 你需要汇报相应的实验结果：演化算法参数设置（种群大小、算子参数、迭代轮数等），问题相关信息（数据集等），画出以评估次数（或迭代轮数）为横轴、fitness 为纵轴的曲线以观察算法的效果

解答:

1.1 算法原理

算法原理如算法1所示。

1.2 算法实现

定义解与种群 本次实验采取二进制表示方法，个体即解采用二进制表示，由于问题为带约束优化的子模优化问题，因此保留的个体 x 包含 1 的个数一般会被限定在 k 附近。由于

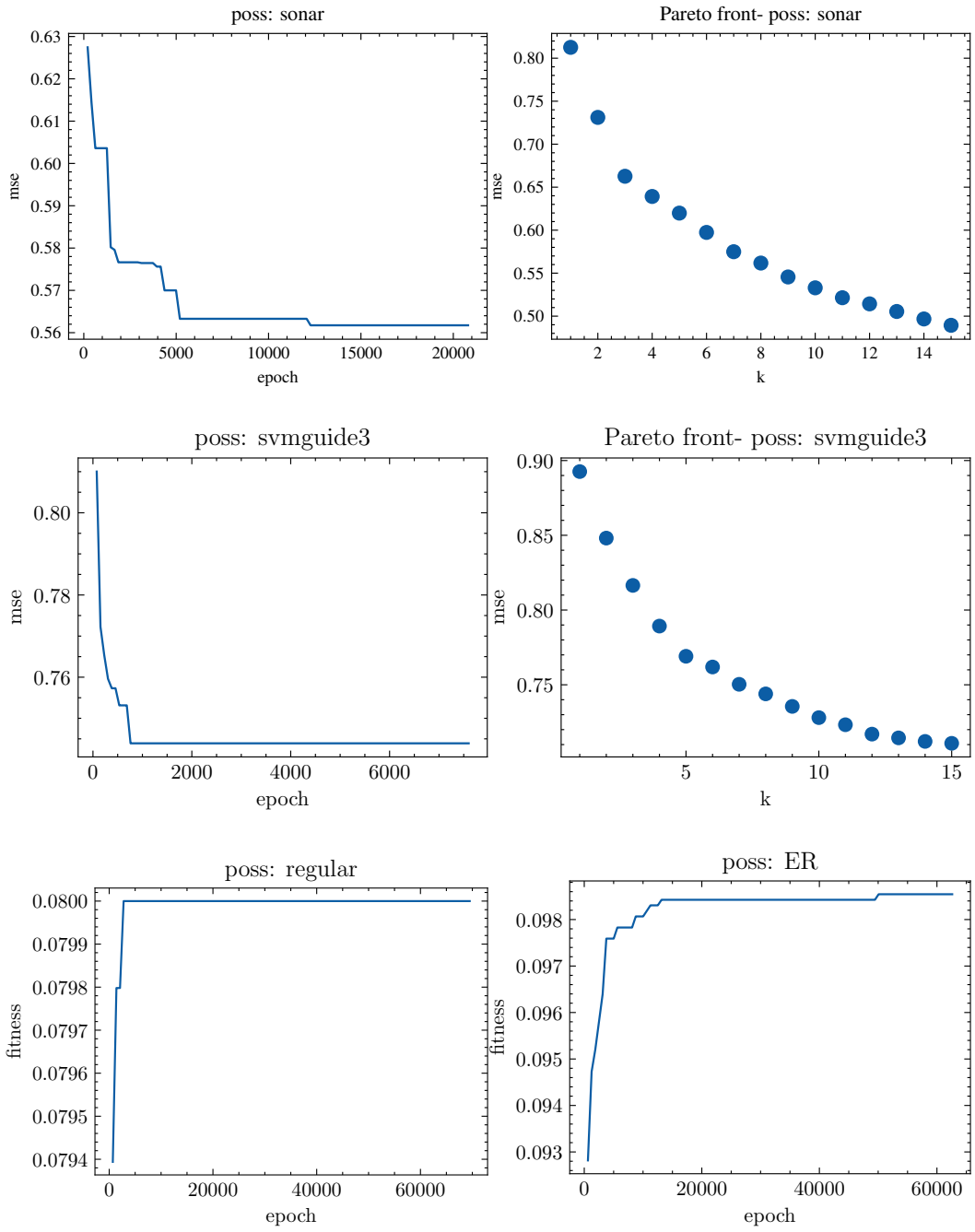

图 1: POSS 算法在稀疏回归和最大覆盖问题的性能图以及最终找到的 Pareto 前沿 (对于稀疏回归问题，记录均方差误差 mse；对于最大覆盖问题，记录标准化的 fitness 值)

算法 1 POSS 算法 [1]

算法输入: 所有变量 $V = \{X_1, ..., X_n\}$，给定目标 f 以及正整数 $k \in [1, n]$
算法参数: 循环轮数 T
算法过程:
$s = \{0\}^n, P = \{s\}$
$t = 0$
while $t < T$ **do**
  从 P 中均匀随机地选择一个解 s.
  通过对 s 的每一位以 $1/n$ 的概率翻转 (若当前值为 0，则变成 1；反之亦然), 产生一个新解 s'.
  if $\nexists z \in P$ 满足 $z \prec s'$ **then**
    $Q = \{z \in P \mid s' \preceq z\}$.
    $P = (P \backslash Q) \cup \{s'\}$.
  end if
  $t = t + 1$
end while
算法输出: $\arg\min_{s \in P, |s| \leqslant k} f_1(s)$

POSS 算法的策略，种群的初始化规模设置为 1，个体为全零向量，即 $s = \{0\}^n$。

设计演化算子 演化算子包含：亲代选择算子、变异算子、交叉算子、生存选择算子。根据 POSS 算法的策略，亲代选择算子为均匀选择算子、变异算子为 bit-wise 变异算子、不使用交叉算子、生存选择算子为仅保留支配占优解。

1.3 实验结果

基础演化算法参数设置如下：种群大小初始大小为 1，迭代中保留所有支配占优解，迭代次数为 $2ek^2n$ 轮，亲代选择算子为均匀选择算子、变异算子为 bit-wise 变异算子、不使用交叉算子、生存选择算子为仅保留支配占优解。

稀疏回归问题的数据集如表1所示，最大覆盖问题的数据集如表2所示。

dataset	#inst	#feat	dataset	#inst	#feat
sonar	208	60	svmguide3	1284	**22**

表 1: 稀疏回归问题的数据集

dataset	节点数	边数	dataset	节点数	边数
正则图	200	9900	ER 图	180	8382

表 2: 最大覆盖问题的数据集

迭代次数-fitness/迭代次数-mse 图像如图1所示，最终收敛的 mse 值/fitness 值如表3所示。(对于稀疏回归问题，记录均方差误差 mse；对于最大覆盖问题，记录标准化的 fitness 值，即割边权重之和除以边数)

2 NSGA-II 与 MOEA/D 算法

任务二 (40pts)

实现 NSGA-II 与 MOEA/D 算法, 用于求解子集选择问题
- 你需要实现两种多目标演化算法：定义解与种群、设计演化算子
- 你需要汇报相应的实验结果，同任务一：将三种方法置于同一张收敛曲线图中比较

dataset	mse	dataset	mse
sonar	0.5617	svmguide3	0.7439
dataset	fitness	dataset	fitness
正则图	0.0800	ER 图	0.0984

表 3: POSS 算法性能

解答:

2.1 算法原理

NSGA-II 算法 [2] NSGA-II 是 2002 年 Deb 等人对其算法 NSGA 的改进, 它是迄今为止最优秀的进化多目标优化算法之一. 相对于 NSGA 而言, NSGA-II 具有以下优点:

(1) 新的基于分级的快速非支配解排序方法将计算复杂度由 $O\left(mN^3\right)$ 降到 $O\left(mN^2\right)$, 其中, m 表示目标函数的数目、N 表示种群中个体的数目.

(2) 为了标定快速非支配排序后同级中不同元素的适应度值, 同时使当前 Pareto 前沿面中的个体能够扩展到整个 Pareto 前沿面, 并尽可能地均匀遍布. 该算法提出了拥挤距离的概念, 采用拥挤距离比较算子代替 NSGA 中的适值度共享方法, 拥挤距离的时间复杂度为 $O(m(2N)\log(2N))$.

(3) 引入了精英保留机制, 经选择后参加繁殖的个体所产生的后代与其父代个体共同竞争来产生下一代种群, 因此有利于保持优良的个体, 提高种群的整体进化水平.

MOEA/D 算法 [3] 将多目标优化问题被转化为一系列单目标优化子问题，然后利用一定数量相邻问题的信息，采用进化算法对这些子问题同时进行优化。在迭代的过程中，当前子问题的当前解成为目前最优的解，可以更新当前子问题的解，同时对邻域也有参考价值。这是因为他们很接近，问题相近，解也大概率相同。MOEA/D 算法的优点如下:

(1) MOEA/D 没有直接将 MOP 作为一个整体来解决, 而是同时优化 N 个标量化的最优化问题, 这样就可以很容易地在 MOEA/D 框架下使用其他的进化算法。

(2) MOEA/D 具有比 NSGA-II 更低的计算复杂度。

(3) 使用更先进的分解方法, MOEA/D 的算法性能将会得到提升等。

算法 2 MOEA/D 算法 [3]

算法输入: 多目标优化, 停止标准, N: MOEA/D 考虑的子问题的数量, N 个权重向量的均匀分布: $\lambda_1, ..., \lambda_N$, T: 每个权重向量的附近的权重向量的数量
算法参数: EP
算法过程:
1: 创建一个外部种群（EP）用于存储优秀个体，初始为空
2: 计算任意两个权重向量之间的欧氏距离，对每个权向量计算与其最近的 T 个权重向量。对于每个 $i = 1, ..., N$, 设置 $B(i) = i_1, ..., i_T$, 其中 $\lambda_{i_1}, , .., \lambda_{i_T}$ 是 λ_i 的最近 T 个权重向量
3: 对于每一个 $i = 1, 2, ..., N$，分别评价 $FV^i = F(x^i)$
4: 根据不同的问题设置 $z = (z_1, z_2, ..., z_m)^T$
5: **for** $i = 1, ..., N$ **do**
6: 再生后代：从 $B(i)$ 中选择 $i = k$ 和 $i = l$ 的两个 x_i 和 x_l 的个体，使用遗传算法生成一个子代解 x'
7: 修正：通过对特殊问题使用修改或启发式的方法对 x' 进行修复
8: 函数评价：评价函数 $F(x')$
9: 更新 z：对于每一个 $j = 1, ..., m$，如果 $z_j > f_j(x')$，将 $z_j = f_j(x')$
10: 替换/更新解：对于每一个 $j \in B(i)$，如果 $g^{TCH}(x'|\lambda_j, z) \leq g^{TCH}(x_j|\lambda_j, z)$，那么将做如下改变：$x_j = x'$, $FV^j = F(x')$
11: 更新 EP：将所有被 $F(x')$ 支配的向量移出 EP，将没有可以支配 $F(x')$ 的向量移入 EP
12: **end for**
13: 若停止条件满足，则算法停止，输出 EP；否则，继续转向第 5 步
算法输出: EP

2.2 算法实现

NSGA-II 算法的核心代码主要包括基于分级的快速非支配解排序方法和拥挤距离函数，前者的主要步骤为: (1) 找到种群中所有 $n_p = 0$ 的个体，并保存在当前集合 F_1 中；(2) 对于当前集合 F_1 中的每个个体 i，其所支配的个体集合为 S_i，遍历 S_i 中的每个个体 l，执行 $n_l = n_l - 1$，如果 $n_l = 0$ 则将个体 l 保存在集合 H 中; (3) 记 F_1 中得到的个体为第一个非支配层的个体，并以 H 作为当前集合，重复上述操作，直到整个种群被分级。

具体代码实现如下所示。

```python
# 快速非支配解排序方法
def fast_non_dominated_sort(values1, values2):
    length = len(values1)
    S = [[] for _ in range(length)]
    front = [[]]
    n = [0 for _ in range(length)]
    rank = [0 for _ in range(length)]

    for p in range(length):
        S[p], n[p] = [], 0 # the set of solutions dominated by p, the number of solutions dominating p
        for q in range(length):
            # if p dominates q, add q to S[p]
            if (values1[p] < values1[q] and values2[p] < values2[q]) or \
                (values1[p] <= values1[q] and values2[p] < values2[q]) or \
                    (values1[p] < values1[q] and values2[p] <= values2[q]):
                if q not in S[p]:
                    S[p].append(q)
            # if q dominates p, increase n[p]
            elif (values1[q] < values1[p] and values2[q] < values2[p]) or\
                (values1[q] <= values1[p] and values2[q] < values2[p]) or\
                    (values1[q] < values1[p] and values2[q] <= values2[p]):
                n[p] += 1

        # p is ranked by 1
        if n[p] == 0:
            rank[p] = 0
            if p not in front[0]:
                front[0].append(p)

    i = 0
    while front[i] != []:
        # store the solutions with the next rank
        Q = []
        for p in front[i]:
            for q in S[p]:
                n[q] -= 1 # As p is excluded now, decrease n[q]
                if n[q] == 0:
                    rank[q] = i + 1
                    if q not in Q:
                        Q.append(q) # q has the next rank
        i += 1
        front.append(Q)
    del front[-1]
    return front
```

```python
# 拥挤距离计算
def crowding_distance(values1, values2, front):
    length = len(front)
    distance = [0 for _ in range(length)]
    sorted1 = sorted(range(length), key = lambda idx: values1[idx])
    sorted2 = sorted(range(length), key = lambda idx: values2[idx])
    distance[0] = np.inf
    distance[length - 1] = np.inf
    gap1 = max(values1) - min(values1)
    gap2 = max(values2) - min(values2)
    for idx in range(1, length - 1):
        distance[idx] += (values1[sorted1[idx + 1]] - values2[sorted1[idx - 1]]) / (gap1 + 1e-4) + \
            (values1[sorted2[idx + 1]] - values2[sorted2[idx - 1]]) / (gap2 + 1e-4)
    return distance
```

MOEA/D 算法用于分解的初始权重向量的生成对于实验效果有很大影响，用于生成高维空间中均匀权重向量的代码如下所示：

```python
# 生成权重向量
def gen_mean_vector(N, m):
    def _distribution_number(sum, m):
        if m == 1:
            return [[sum]]
        vectors = []
        for i in range(1, sum - (m - 1) + 1):
            right_vec = _distribution_number(sum - i, m - 1)
            vectors.extend([i] + item for item in right_vec)
        return vectors

    vectors = _distribution_number(N + m, m)
    vectors = (np.array(vectors) - 1) / N
    return vectors, len(vectors)
```

2.3 实验结果

定义解与种群 本次实验采取二进制表示方法，个体即解采用二进制表示，由于问题为带约束优化的子模优化问题，因此初始化时个体 x 包含 1 的个数一般会被限定在 k 附近，即解 x 的每个维度都有 k/n 的可能等于 1。

NSGA-II 算法的种群初始化规模设置为 20。由于 MOEA/D 算法将多目标问题分解一系列单目标优化子问题，因此初始权重向量需要设置的稠密一些，初始权重向量个数为 100，相应的种群规模设置为 $C_{100}^2 = 101$。

设计演化算子 演化算子包含：亲代选择算子、变异算子、交叉算子、生存选择算子。根据 NSGA-II 算法的策略，亲代选择算子为均匀选择算子、变异算子为 bit-wise 变异算子、交叉算子为单点交叉算子、生存选择算子为 N+N 选择算子。MOEA/D 算法基于分解策略实现，因此算子设置与其他方法略有不同，亲代选择和生存选择都仅在相邻子问题中产生，变异算子为 bit-wise 变异算子、交叉算子为单点交叉算子，最终最优解集的保留策略为仅保留支配占优解。

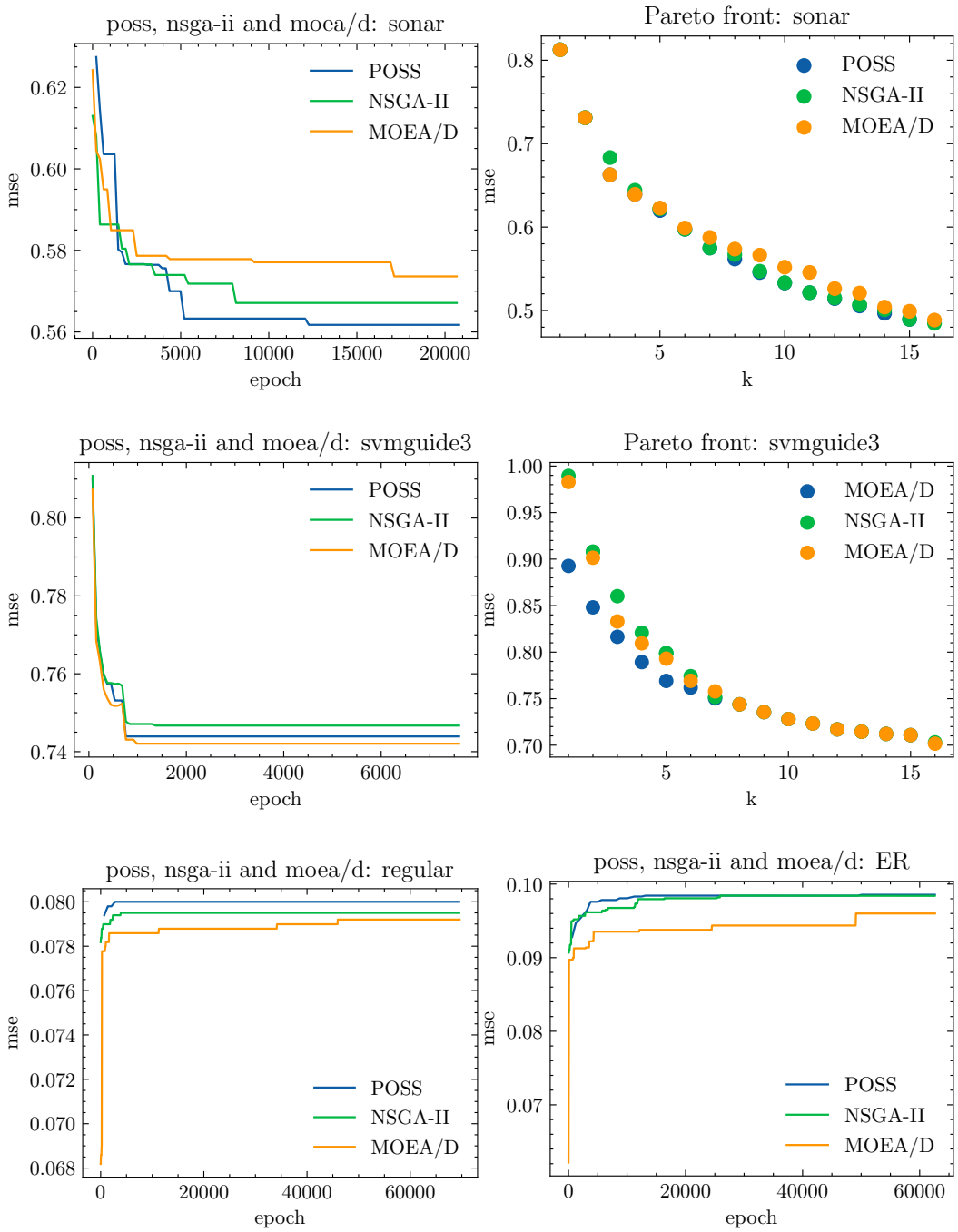

图 2: NSGA-II、MOEA/D 算法在稀疏回归和最大覆盖问题的性能图以及最终找到的 Pareto 前沿 (对于稀疏回归问题，记录均方差误差 mse；对于最大覆盖问题，记录标准化的 fitness 值)

实验结果 基础演化算法参数设置如下：NSGA-II 算法的种群大小初始大小为 20，迭代次数为 $2ek^2n$ 轮，MOEA/D 算法的种群大小初始大小为 101，迭代次数为 $2ek^2n$ 轮，两种方法的演化算子设计如上文所述。

稀疏回归问题的数据集如表1所示，最大覆盖问题的数据集如表2所示。

迭代次数-fitness/迭代次数-mse 图像如图2所示，最终收敛的 mse 值/fitness 值如表4所示。(对于稀疏回归问题，记录均方差误差 mse；对于最大覆盖问题，记录标准化的 fitness 值，即割边权重之和除以边数)

根据图2可以发现，可能是由于 NSGA-II 和 MOEA/D 代码实现的原因，POSS 算法的收敛性能优于 NSGA-II 和 MOEA/D 算法，并且在最大覆盖问题上，由于采用了正则图和 ER 图，POSS 算法可以达到理论的最优解。而 NSFA-II 算法得到的 Pareto 前沿较为均匀，MOEA/D 算法由于受分解策略的影响较大，性能略逊于其他算法。

algorithm	dataset	mse	dataset	mse	dataset	fitness	dataset	fitness
POSS	sonar	**0.5617**	svmguide3	0.7439	正则图	**0.0800**	ER 图	**0.0984**
NSGA-II	sonar	0.5671	svmguide3	0.7466	正则图	0.0795	ER 图	**0.0984**
MOEA/D	sonar	0.5736	svmguide3	**0.7420**	正则图	0.0792	ER 图	0.0960

表 4: NSGA-II、MOEA/D 算法性能

3 改进算法

任务三 (20pts)

算法改进：设计更适合子集选择问题的算法
- 可能的改进方向：问题建模、多目标优化算法改进
- 如果参考了论文，请规范引用并标明
- 你需要汇报相应的实验结果，同任务一和任务二：将四种方法置于同一张收敛曲线图中比较

解答：

3.1 算法原理

基于分解策略进行改进。本思路基于参考文献 [4]。POSS 算法的计算复杂度为 $2enk^2$，仍存在可以加速的空间。即可以在 POSS 算法的基础上，通过限定候选集 P 的大小，多次运行 POSS 算法去逐段找到中间解，而不再是只运行一次 POSS 算法直接去找到所有中间解。根据参考文献 [4] 的理论证明，改进后的算法在获得与 POSS 算法相同近似性能下界的同时，运行时间随着分解个数的增加超线性下降，迭代总次数约为 $2ekn\lceil\frac{k}{m}\rceil$。

3.2 算法实现

定义解与种群 本次实验采取二进制表示方法，个体即解采用二进制表示，由于问题为带约束优化的子模优化问题，因此保留的个体 x 包含 1 的个数一般会被限定在 k 附近。由于 DPOSS 算法的策略，首次运行的种群初始化规模设置为 1，个体为全零向量，非首次运行时使用上一次运行产生的种群进行初始化。

算法 3 DPOSS 算法 [4]

算法输入: 所有变量 $V = \{X_1, ..., X_n\}$，给定目标 f 以及正整数 $k \in [1, n]$
算法参数: 分解个数 $m \in [1, k]$，各阶段 POSS 方法运行轮数 $T_1, T_2, ..., T_m$
算法过程:
$i = 1$
while $i <= m$ **do**
  if i=1 **then**
    使用 POSS 方法去找解集 $P_1^\star$，其中初始解为 $\{0\}^n$，运行轮数为 T_1
    在 POSS 方法终止运行后，输出解为 $s_{k_1}^\star = \arg\min_{s \in P, |s| \le k_1} f_1(s)$
  else
    使用 POSS 方法去找解集 $P_i^\star$，其中初始解为 $s_{k_{i-1}}^\star$，运行轮数为 T_i
    在 POSS 方法终止运行后，输出解为 $s_{k_i}^\star = \arg\min_{s \in P, |s| \le k_i} f_1(s)$
  end if
  $i = i + 1$
end while
算法输出: $s_{k_m}^\star$

设计演化算子 演化算子包含：亲代选择算子、变异算子、交叉算子、生存选择算子。根据 DPOSS 算法的策略，亲代选择算子为均匀选择算子、变异算子为 bit-wise 变异算子、不使用交叉算子、生存选择算子为仅保留支配占优解。并且在实验中，方便起见，设置轮数为 k 轮，因此迭代总次数约为 $2ekn$。

3.3 实验结果

基础演化算法参数设置如下：首次运行的种群初始化规模设置为 1，非首次运行时使用上一次运行产生的种群进行初始化，迭代中保留所有支配占优解，迭代总次数约为 $2ekn\lceil \frac{n}{m} \rceil$。

稀疏回归问题的数据集如表1所示，最大覆盖问题的数据集如表2所示。

迭代次数-fitness/迭代次数-mse 图像如图3所示，最终收敛的 mse 值/fitness 值如表5所示。(对于稀疏回归问题，记录均方差误差 mse；对于最大覆盖问题，记录标准化的 fitness 值，即割边权重之和除以边数)

根据图3可以发现，DPOSS 算法通过限定候选集 P 的大小，多次运行 POSS 算法去逐段找到中间解，**在获得与 POSS 算法相同近似性能下界的同时，运行时间随着分解个数的增加超线性下降**，迭代总次数由 $2enk^2$ 变为 $2ekn\lceil \frac{k}{m} \rceil|_{m=k} = 2ekn$ 。

algorithm	dataset	mse	dataset	mse	dataset	fitness	dataset	fitness
POSS	sonar	**0.5617**	svmguide3	0.7439	正则图	**0.0800**	ER 图	**0.0984**
NSGA-II	sonar	0.5671	svmguide3	0.7466	正则图	0.0795	ER 图	**0.0984**
MOEA/D	sonar	0.5736	svmguide3	**0.7420**	正则图	0.0792	ER 图	0.0960
改进算法	sonar	0.5652	svmguide3	0.7442	正则图	**0.0800**	ER 图	0.0983

表 5: 改进算法性能

参考文献

[1] Q. Chao, Y. Yu, and Z. H. Zhou. Subset selection by pareto optimization. In *Advances in Neural Information Processing Systems 28 (NIPS'15)*, 2015.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

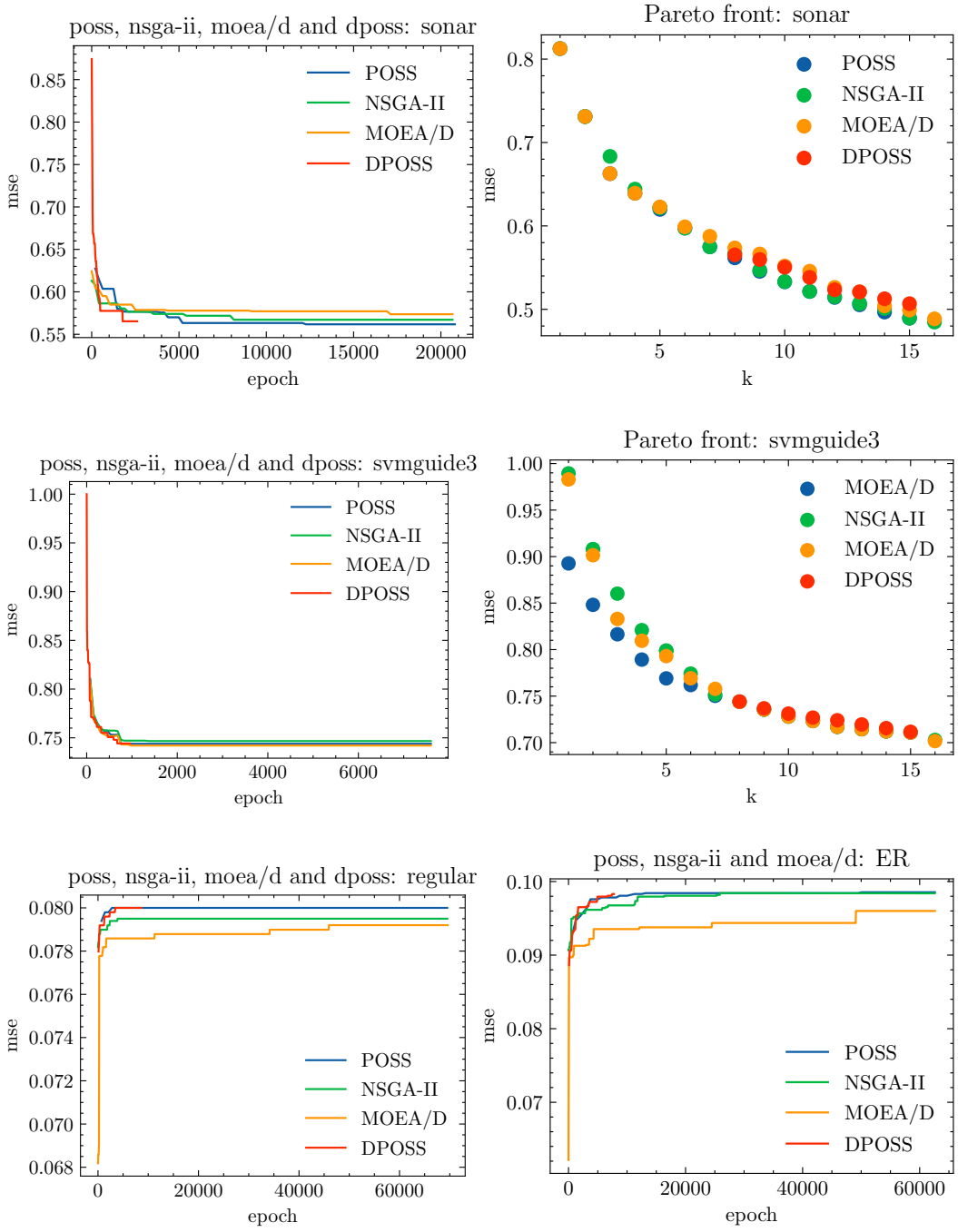

图 3: 改进算法在稀疏回归和最大覆盖问题的性能图以及最终找到的 Pareto 前沿 (对于稀疏回归问题，记录均方差误差 mse；对于最大覆盖问题，记录标准化的 fitness 值)

[3] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. In *IEEE International Conference on Advanced Learning Technologies*, 2005.

[4] 钱超, 周志华. 基于分解策略的多目标演化子集选择算法. 中国科学: 信息科学, 46(9):1276, 2016.