HSEA-2022 Homework4

201300096 人工智能学院 杜兴豪

具体实现

接下来的实现中，所采用的数据集都是一致的，具体见最后的数据集介绍部分。

问题与解的定义

在本次实现中，我将问题分别封装到了对应的文件中：

- 稀疏回归问题：sparseRegression.py
- 最大覆盖问题：maxCoverage.py

在对应的文件中，我分别对问题的两个目标进行了封装，对应接口均为`obj1(solution)`和`obj2(solution)`，在演化算法的实现过程中，仅需要导入对应文件，并引用其问题函数即可。

关于解的定义，由于本次的优化目标为子集选择问题，因此将集合中的所有元素用二进制串来表示，串值为0表示不选择该元素，串值为1表示选择，这样的方式即为很简洁易操作的了。

此外，问题、解与种群的定义在各种演化算法中都是一致的。种群的初始定义也是在对应文件中完成，因此在演化算法中并不需要知道解的具体长度，只需调用对应问题下的`Population`全局变量即可，并且它们的初始化值均为全0串，也保证了演化算法的正确性和收敛性。

POSS

算法过程为每次将生成的一个子代解与原种群中的所有解进行对照，如果这个子代解不被种群中任意解dominate住，则将其加入种群，并删掉种群中所有被它dominate住的解。以此类推，直到完成所有迭代轮数，从种群中选出最优解。在稀疏回归中表示为满足不等式约束$|S| \leq k$的具有最小MSE的解。

Algorithm 2 POSS

Input: all variables $V = \{X_1, \ldots, X_n\}$, a given criterion f and an integer parameter $k \in [1, n]$
Parameter: the number of iterations T and an isolation function $I : \{0,1\}^n \to \mathbb{R}$
Output: a subset of V with at most k variables
Process:

Let $\boldsymbol{s} = \{0\}^n$ and $P = \{\boldsymbol{s}\}$.
Let $t = 0$.
while $t < T$ **do**
  Select $\boldsymbol{s}$ from P uniformly at random.
  Generate $\boldsymbol{s}'$ from $\boldsymbol{s}$ by flipping each bit of $\boldsymbol{s}$ with probability $1/n$.
  if $\nexists \boldsymbol{z} \in P$ such that $I(\boldsymbol{z}) = I(\boldsymbol{s}')$ and $\big((\boldsymbol{z}.o_1 < \boldsymbol{s}'.o_1 \wedge \boldsymbol{z}.o_2 \leq \boldsymbol{s}'.o_2)$ or $(\boldsymbol{z}.o_1 \leq \boldsymbol{s}'.o_1 \wedge \boldsymbol{z}.o_2 < \boldsymbol{s}'.o_2)\big)$ **then**
    $Q = \{\boldsymbol{z} \in P \mid I(\boldsymbol{z}) = I(\boldsymbol{s}') \wedge \boldsymbol{s}'.o_1 \leq \boldsymbol{z}.o_1 \wedge \boldsymbol{s}'.o_2 \leq \boldsymbol{z}.o_2\}$.
    $P = (P \setminus Q) \cup \{\boldsymbol{s}'\}$.
  end if
  $t = t + 1$.
end while
return $\arg\min_{\boldsymbol{s} \in P, |\boldsymbol{s}| \leq k} f(\boldsymbol{s})$

代码实现方案严格按照论文中提到的伪代码实现过程。注意到伪代码中并未限制种群大小，因此我实现的POSS算法中也将限制种群大小的代码做了注释，具体为

```
if len(Population) > PopulationUpperBound:
    RankCount(Population)
    sortbyRank(Population)
    Population = Population[:PopulationUpperBound]
```

即调用在NSGAII中用到的秩排序方法，对种群排序后，将rank较低的一些个体筛除。如果取消代码中的这段注释，可能获得更好的表现，让MSE收敛更快，但是这就不是纯粹的POSS算法了。

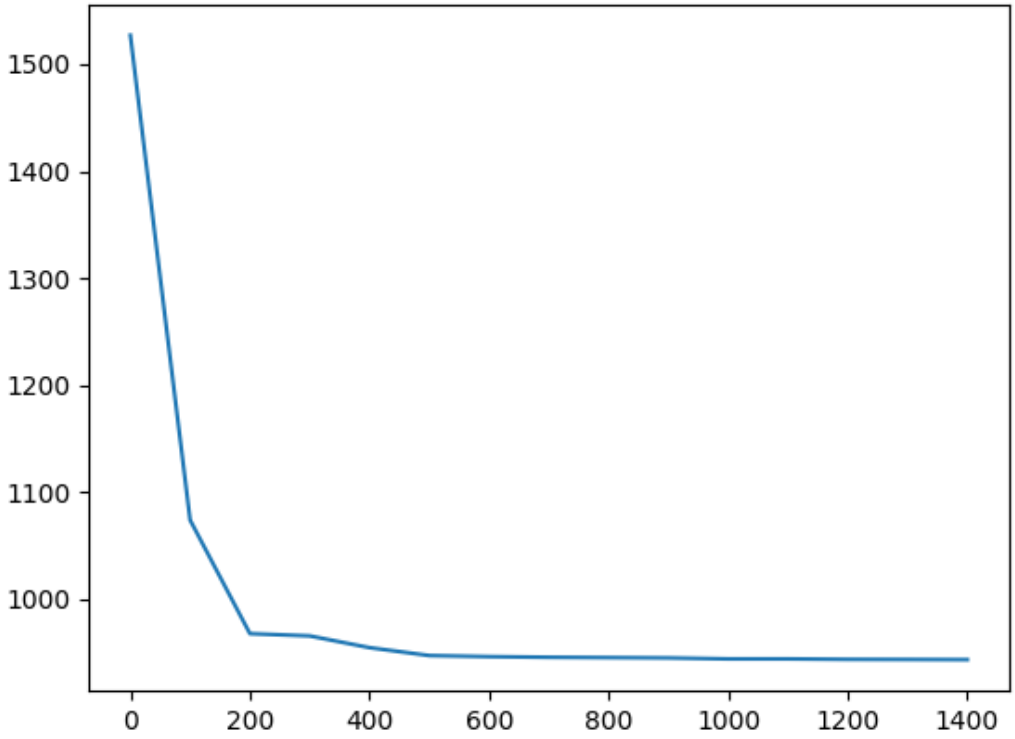
实现结果

对于稀疏回归问题

实验中采用了这样的参数设置：

- 种群规模：无（POSS算法不设种群规模上限）
- 最大选取属性个数（即问题的K值）：60
- 迭代轮数：从0到1500

最终结果：



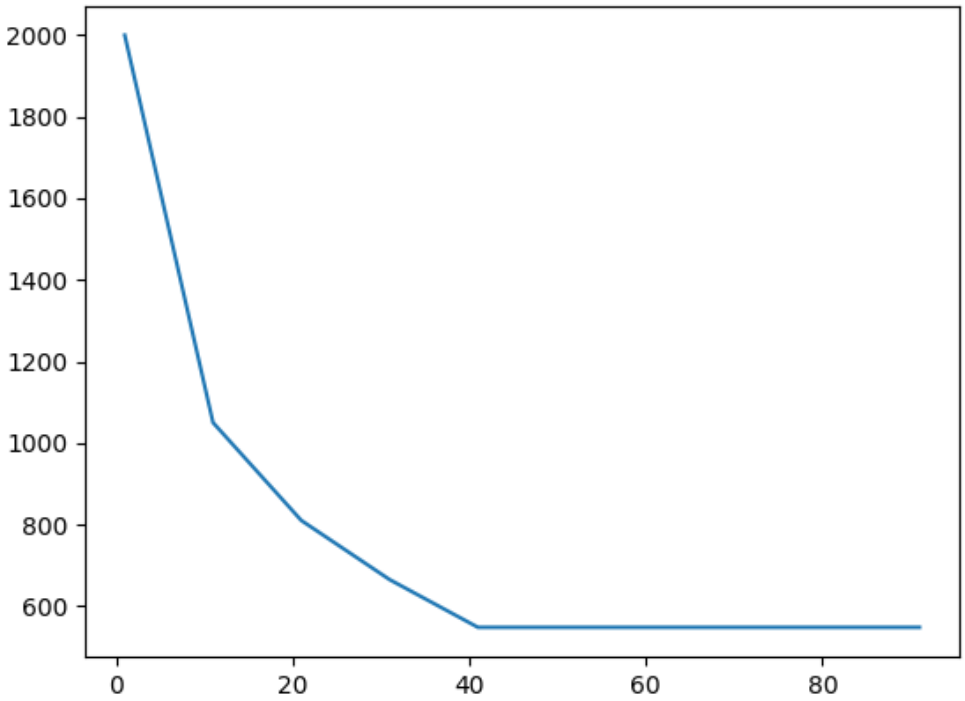
最终的MSE降到了940左右。和NSGAII对照后发现，这个值合理。

对于最大覆盖问题：

参数设置：

- 图中顶点总数：2000
- 顶点集数目：100
- 最多可选择的顶点集数目：20

- 迭代轮数：0-100（该问题收敛速度很快）



NSGAII

该算法严格按照演化算法的流程，具体包含以下部分：

- 初始化种群：我采用全0串充满整个种群作为初始种群。
- 父代选择：Binary tournament selection。设置一个全局字典Rank。每次在选择父代之前，对种群进行non-dominated sorting，结果保存在Rank字典中。选择时通过Rank进行最优筛选。
- 变异和交叉互换：我选用单点交叉，任意选择一个位置，从两个父代的相同位置交叉互换，生成两个初始的子代解。再通过均值为1的均匀变异，改变每个子代解，生成最终的子代，放入种群。
- 生存选择：按照non-dominated sorting下的结果，选择rank最优的前N个解，维持种群规模不变。

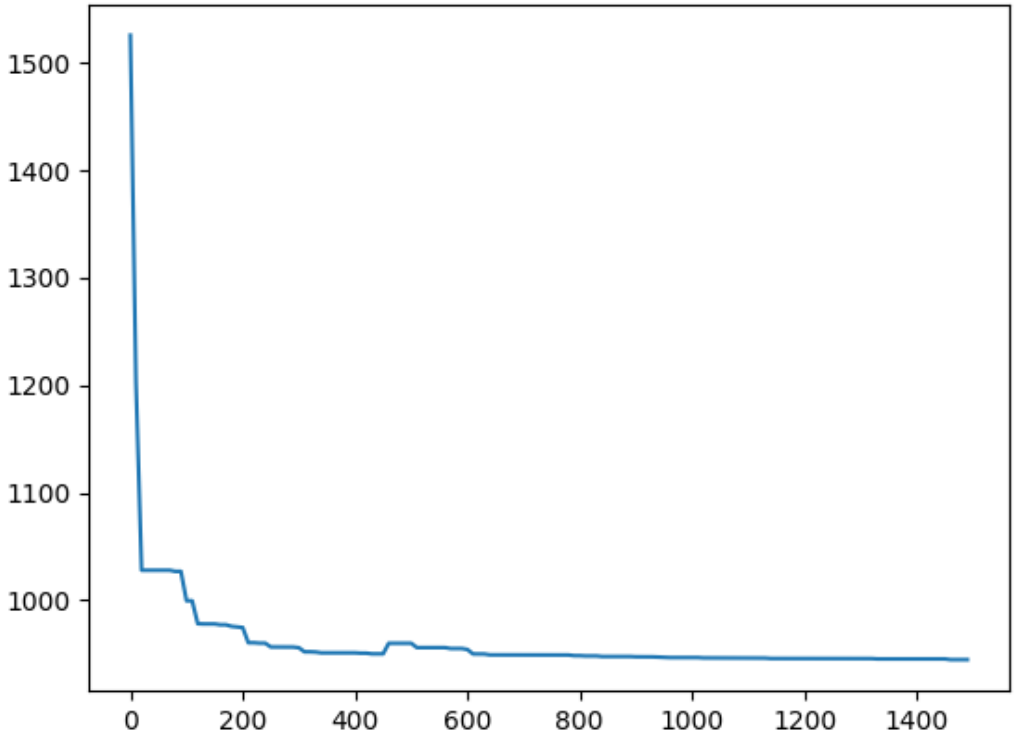
实现结果

稀疏回归

涉及的参数设置：

- 种群规模：20
- 父代选择时tournament selection的子种群大小：20
- 最大选择的属性个数（即问题的K值）：60

结果的epoch-fitness图：

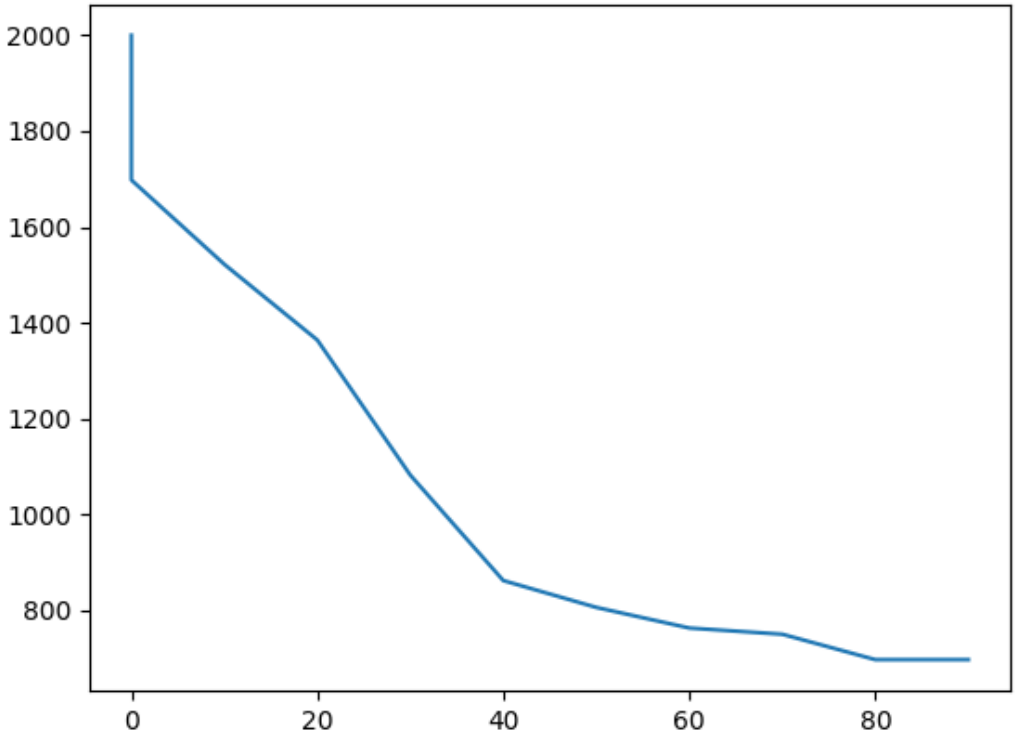


最大覆盖

参数设置：

- 图中顶点总数：2000
- 顶点集数目：100
- 最多可选择的顶点集数目：20
- 种群大小：20
- 最大迭代轮数：100（该问题收敛速度很快）
- 父代选择时tournament selection的子种群大小：20

结果的epoch-fitness图：



MOEA/D

这个算法的实现思路并不复杂。定义种群规模为N，元素为各个子问题的当前最优解即可。在本次实现中生成的子优化问题为：

$$
\begin{aligned}
&G_i(x) = \lambda_{i1} f_1(x) + \lambda_{i2} f_2(x) \\
&s.t. \quad \lambda_{i2} + \lambda_{i2} = 1 \qquad\qquad\qquad i = 0, 1, \ldots, N-1
\end{aligned}
$$

其中参数设置按照：

$$
\begin{aligned}
\lambda_{i1} &= \frac{i+1}{\mathrm{lenth(x)}} \\
\lambda_{i2} &= 1 - \lambda_{i1}
\end{aligned}
$$

根据问题的定义，当$i = N - 1$时，该问题将仅仅优化子问题1，而不对子问题2设置障碍，即不限制解中覆盖元素的数量。

实现结果

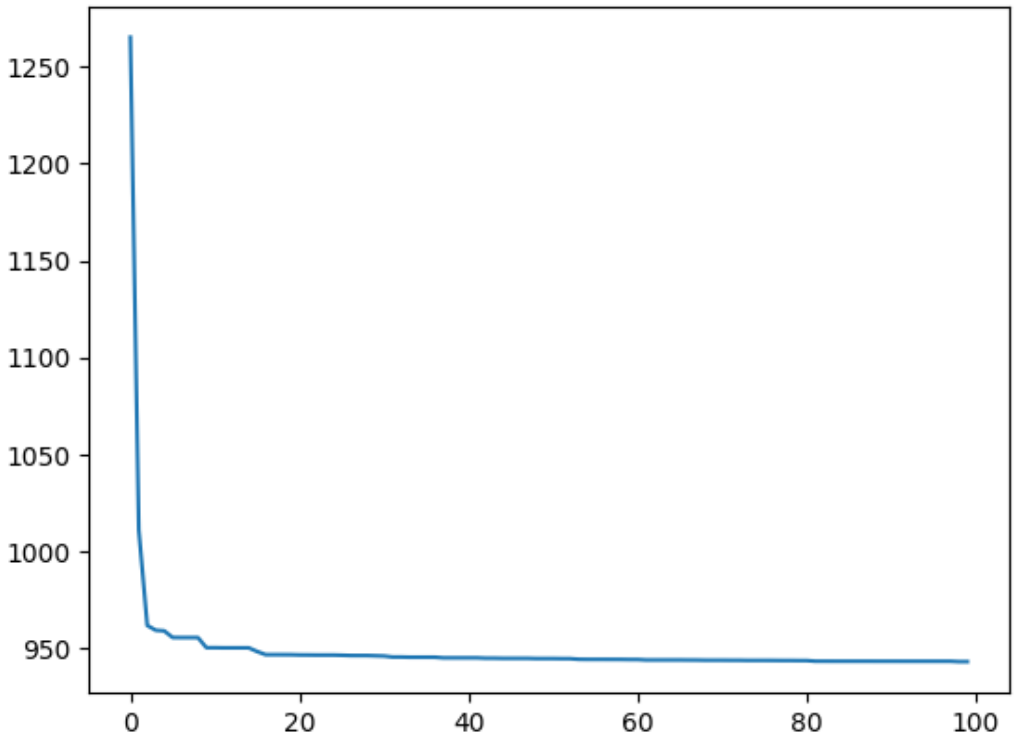
稀疏回归

涉及的参数设置：

- 种群规模（子问题数）：10
- 最大选择的属性个数（即问题的K值）：60

由于该方法下，在每轮评估中会对每一个解都进行更新，实质上近似于其他方法下的$epoch * len(Population)$轮次迭代，因此该方法下解的关于迭代轮数的收敛速度非常快。因此定义迭代轮数上限为100轮，和前几个问题的1000轮一致。

结果的epoch-fitness图：

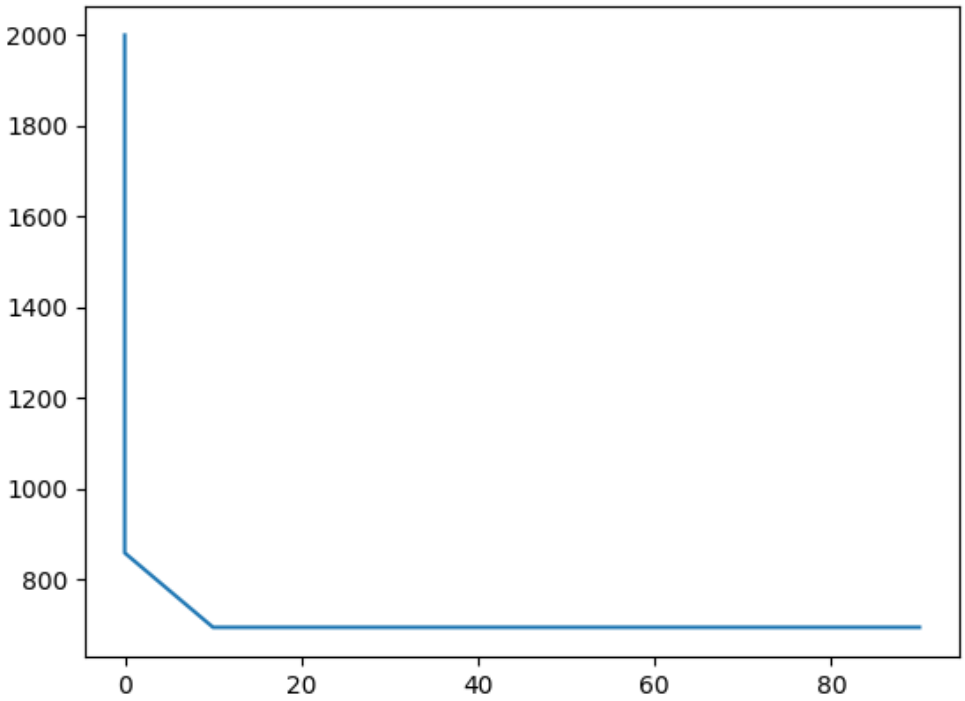


最大覆盖

参数设置：

- 图中顶点总数：2000
- 顶点集数目：100
- 最多可选择的顶点集数目（即问题的K值）：20

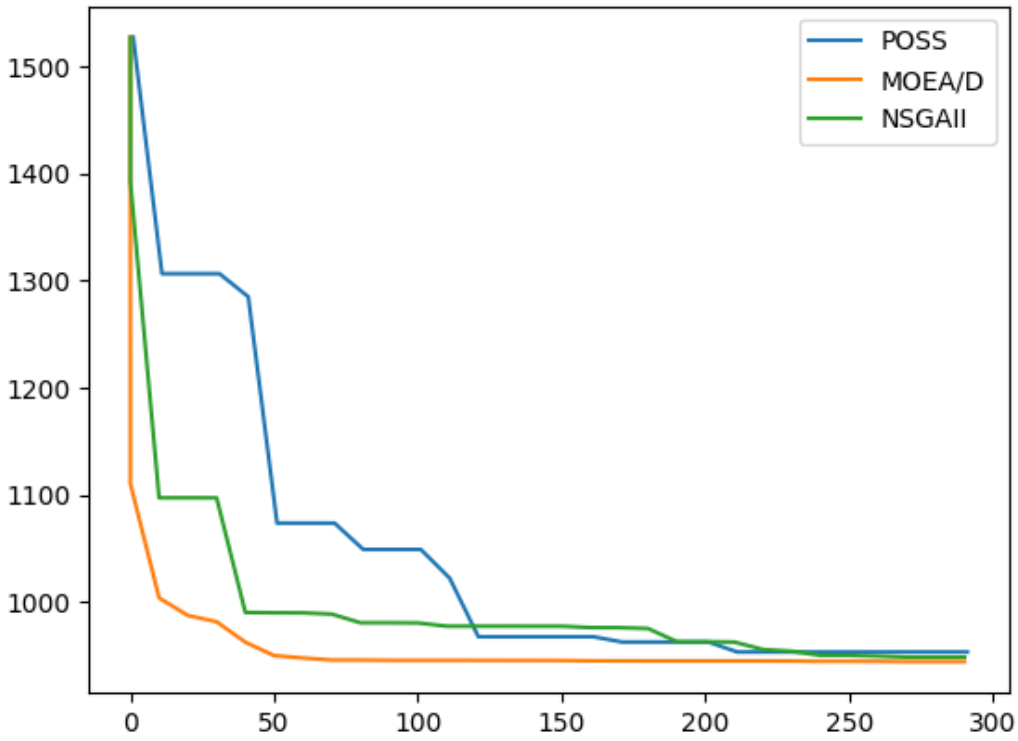
- 种群规模（子问题数）：10



对于该方法而言，每轮迭代中都会涉及传播到邻近解，因此在每一轮迭代中优秀的子代解会像水波一样传递开，导致收敛速度极快。

三种方法汇总对比

通过收敛较慢，迭代轮数较多的稀疏回归数据集，来对比三种方法的实现结果：



观察到MOEA/D的收敛速度最快，其次是NSGAII，最后是POSS算法。然而由于POSS算法不包含种群，其迭代效果取决于某次突变下是否会产生较好的子代解，因此POSS算法不能通过对比epoch-fitness图的方法来判别其优劣。并且，POSS算法得出最优解的速度极快，是这三种（包含我之后实现的新算法）中最快的。

改进算法

改进思路

对于子集选择问题而言，由于该问题的两个目标中，有一个目标仅仅是限制最终选择的解的大小，和另一个问题（比如稀疏回归中选择的属性数，最终反映到拟合程度上）的重要性其实并不一致。

因此我认为，对于该问题来说，我们可以把限制子集数目的这一个优化目标放到演化算子中进行筛选，即我们只在满足解大小的范围内进行演化，这样就可以把原本的问题转化为一个单目标优化问题，只需要把重心放到优化选择对象上，这样或许可以得到更高的效率。

算法设计

仅仅采用常规的单目标优化算法，区别在子代解生成的过程中，我们需要筛选掉那些不满足约束的解，并重新补上相同数目的子代解，直到所有生成的子代解都满足不等式约束。为了生成足够多的子代，并且满足多样性需求，我们同样采用tournament selection来挑选父代解，从选出的部分父代解中每次随机抽样作为父母，来生成新的子代解。

结果展示

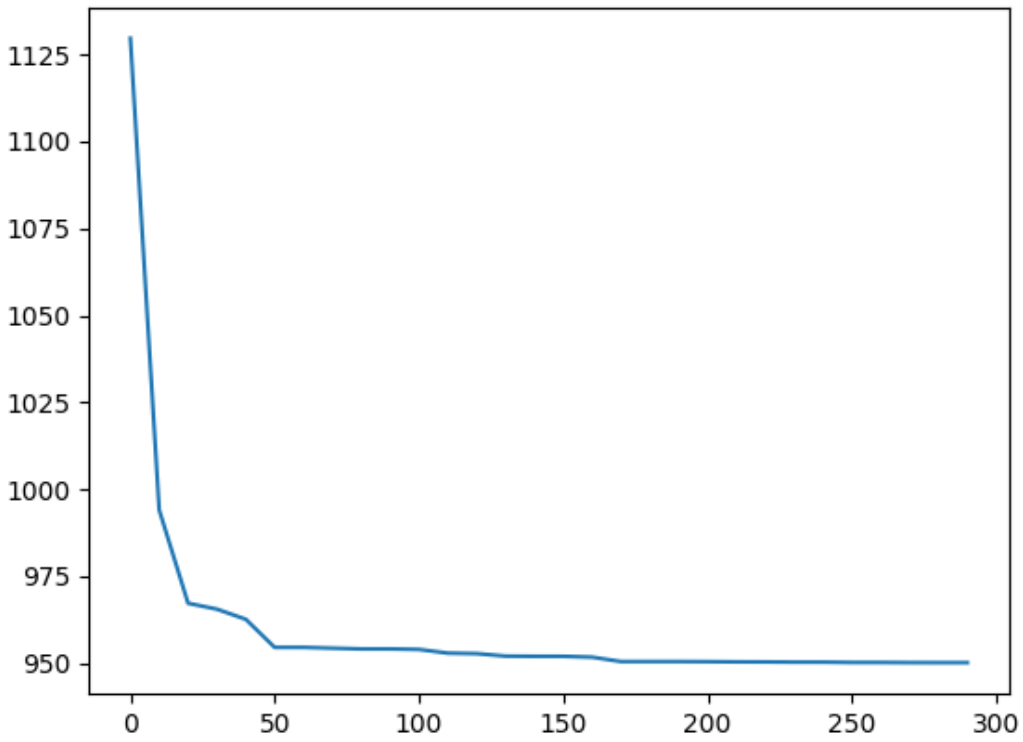
采用的数据集仍然为与前两问一致的数据。

稀疏回归

参数设置：

- 种群规模：10
- Tournament Selection挑选父代解的参数：
  - 挑选时父代解子种群的大小$k = 5$
  - 最终生成的父代解群个数$\lambda = 3$

最终生成的epoch-fitness图：



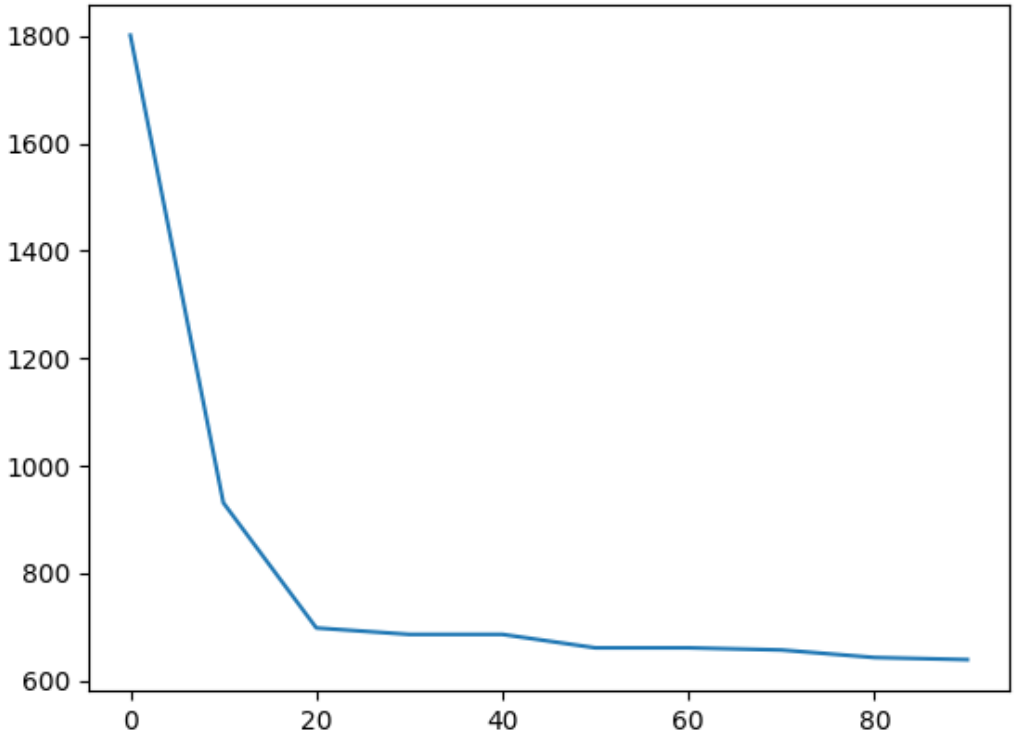
其结果和前三种算法收敛的值一致，证明实现合理。并且我的新算法下降平滑，更具有鲁棒性。

最大覆盖

参数设置：

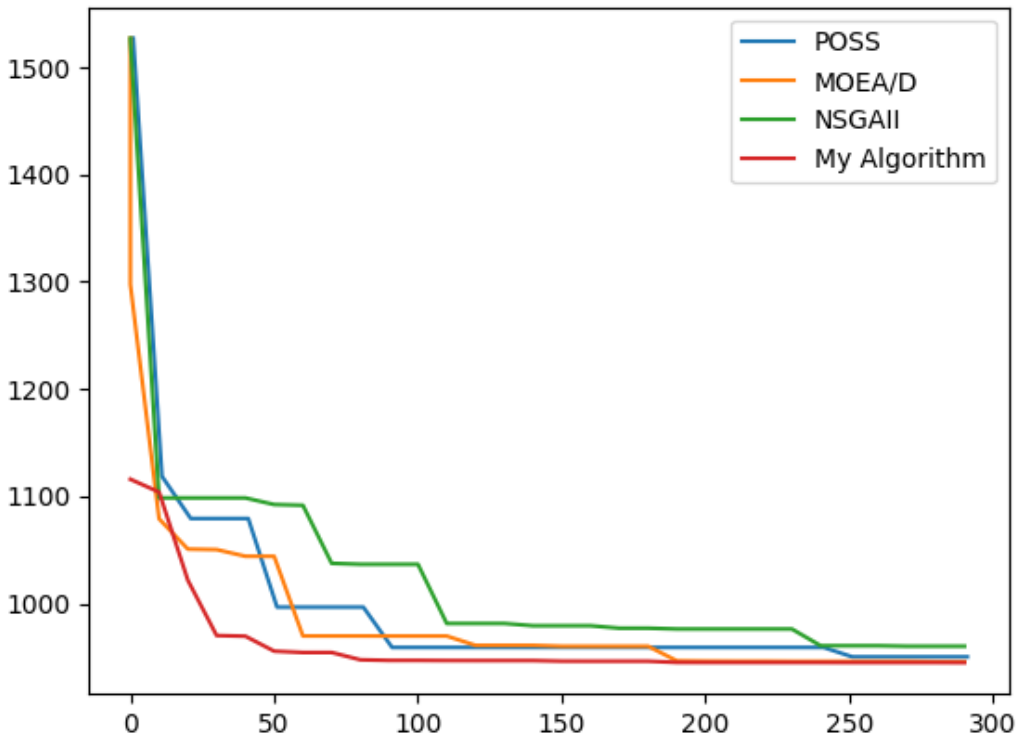
- 图中顶点总数：2000
- 顶点集数目：100
- 最多可选择的顶点集数目（即问题的K值）：20
- 种群规模（子问题数）：10

最终生成的epoch-fitness图：



四种方法汇总对比

算法实现后，选用网站提供的标准数据集，即稀疏回归问题的数据集，展示迭代300轮的过程中，最终的MSE下降情况：



容易看出，我改进后的算法下降效率近似于最快的算法MOEA/D，并且下降过程更为平滑可靠，最终也

得到了最优的情况。

此外，由于删去了不必要的多目标优化过程及其相关的多值排序过程，我的算法的运行速度也有了提升。

数据集

稀疏回归

数据集出处链接：
http://archive.ics.uci.edu/ml/datasets/BlogFeedback
数据来源于用户发布的blog情况，通过各种属性预测用户该条blog在24小时内将获得的评论数。该数据集中含有52397条数据，每条数据含有281个属性，其中最后一个属性为用于回归预测的目标值。

BlogFeedback Data Set

Download: Data Folder, Data Set Description

Abstract: Instances in this dataset contain features extracted from blog posts. The task associated with the data is to predict how many comments the post will receive.

Data Set Characteristics:	Multivariate	**Number of Instances:**	60021	**Area:**	Social
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	281	**Date Donated**	2014-05-29
Associated Tasks:	Regression	**Missing Values?**	N/A	**Number of Web Hits:**	129188

Source:

Krisztian Buza
Budapest University of Technology and Economics
buza '@' cs.bme.hu
http://www.cs.bme.hu/~buza

Data Set Information:

This data originates from blog posts. The raw HTML-documents of the blog posts were crawled and processed.
The prediction task associated with the data is the prediction of the number of comments in the upcoming 24 hours. In order to simulate this situation, we choose a basetime (in the past) and select the blog posts that were published at most 72 hours before the selected base date/time. Then, we calculate all the features of the selected blog posts from the information that was available at the basetime, therefore each instance corresponds to a blog post. The target is the number of comments that the blog post received in the next 24 hours relative to the basetime.

观察到数据集中含有多条重复数据，可能由于数据的重复导致矩阵不可逆（而非属性线性相关），因此我做了数据预处理，将完全重复的数据删去，剩余49203条数据。每条数据中含有的280个属性中，有的几乎为全0，有的几乎完全重复，非常适合用于稀疏回归任务。

csv数据以及其压缩文件保存于blogFeedback文件夹下。

最大覆盖

这项任务的数据集需要我们生成一张图，以及图上的一些点集子集，以便于进行子集覆盖。由于最大覆盖问题并不涉及边的问题，所以没必要真正生成一张完整的图，这里采用规定好点的总个数N和子集的数目k，再随机生成k个子集的方式即可

```
# 创建点集的集合
VerticeSets = []
for i in range(VerticeSetNum):
    currentVerticeSet = []
    for j in range(maxNodeSize):
        if random.random() < zeroProbability:
            currentVerticeSet.append(0)
        else:
            currentVerticeSet.append(1)
    VerticeSets.append(currentVerticeSet)
```

通过上述方法生成的是一个图的列表。列表中的每个元素都是一个由01串组成的图，其中0表示该点不在当前元素所表示的子图中。由此，在之后的度量中，我们知道：

1. 覆盖的面积就是所选的所有子集并后，得到的图中1的总个数
2. 所选子集的个数就是我们从列表`verticeSets`中挑选得到的元素个数。

此外，上述优化目标2同时也可以用01串进行简单表示，对于`verticeSets`的一个01串就是该问题的一个解：0的部分表示该子图不被选择，1的部分表示该子图被选择。因此优化目标2又可以改写为：

- 解01串中1的个数

考虑到随机生成子图时，0和1的概率应为各半，则期望意义下选择两个集合就能够达到覆盖整张图，即最优情况。为了增加迭代过程中的难度，我限制了每个子集中0的个数，即代码中提到的`zeroProbability`，设置为0.85，只有当随机出的概率小于这个值时，才在子图中出现一个0，。保证每个子图的稀疏性，从而清除地展现出迭代演化的过程。

注意到由于我这种数据生成方式是随机生成的，每一次生成的子集选择问题都可能不同，因此我采用的方法为：一次生成数据，同时运行四种算法，通过四种算法之间的实现对比，可以看出其实现效果。

通过上述介绍的过程，我们定义了最大覆盖问题，具体实现在`maxCoverage.py`文件中。

运行方法

获取单个算法的实现结果：

- 运行`POSS.py`文件，可以获得我实现的POSS算法关于问题`problem`的epoch-fitness图
- 运行`NSGAII.py`文件，可以获得我实现的NSGAII算法关于问题`problem`的epoch-fitness图
- 运行`MOEAD.py`文件，可以获得我实现的MOEA/D算法关于问题`problem`的epoch-fitness图
- 运行`NEWALG.py`文件，可以获得我改进后的演化算法关于问题`problem`的epoch-fitness图

若要修改problem所对应的问题，可以更改文件中对于变量`problem`的定义。我初始化为`sparseRegression.py`文件所定义的稀疏回归问题。若要改为最大覆盖问题，只需要在对应的文件中更改以下代码：

```
---  problem = sparseRegression
+++  problem = maxCoverage
```

若要获取四种算法的对比情况，请运行`main.py`文件。其初始化和更改方式与上述描述的过程一致。