

Task 1

Same Stats, Different Graphs---Datasaurus

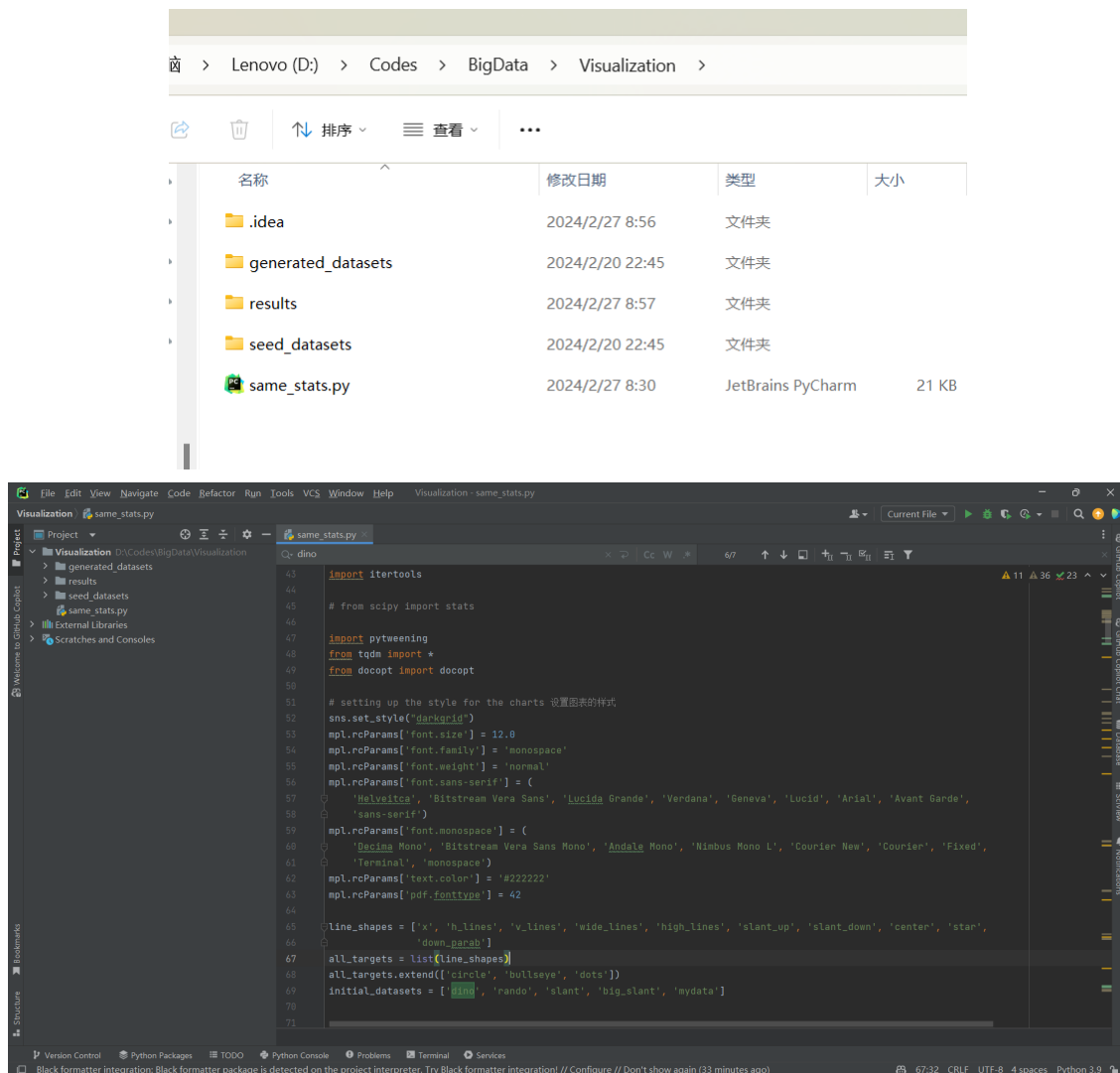
专业：信息安全 学号：2111408 姓名：周钰宸

论文复现

让Datasaurus数据集经过默认次数的扰动，变化为其他形状（比如说circle）

1. 搭建实验环境

使用Pycharm作为IDE进行实验，解决好所需的数据包和对应的依赖



2. 复现过程：











首先使用最基本的命令进行尝试，即使用dinosaur(dino)作为初始形状，**circle**作为目标形状，扰动次数选择200000次迭代扰动，保留2位小数的统计学精度，均匀取其中150张图片保存下来。

```
1 python same_stats.py run dino circle 200000 2 150
```

```
Anaconda Prompt (Anaconda)
(base) C:\Users\zyc13>D:
(base) D:\>cd D:\Codes\BigData\Visualization
(base) D:\Codes\BigData\Visualization>python same_stats.py run dino circle 200000 2 150
circle pattern: 100%##### 200001/200001 [03:46<00:00, 882.22it/s]
(base) D:\Codes\BigData\Visualization>_
```

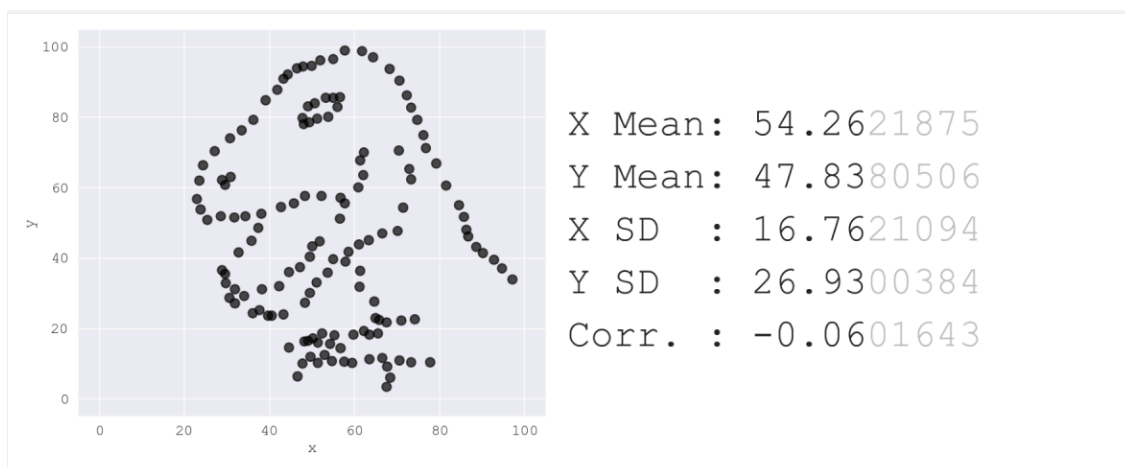
3. 复现结果：查看results文件夹。

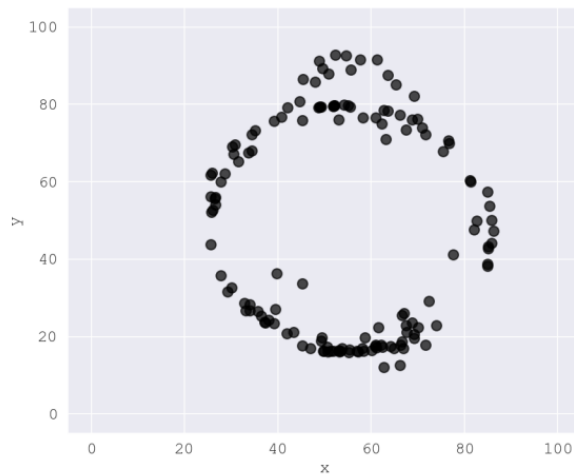
可以看到生成了149个记录circle个data的csv文件，和145个中间过程图像。

名称	修改日期	类型	大小
 circle-data-00146.csv	2024/2/27 9:21	Microsoft Excel ...	6 KB
 circle-data-00147.csv	2024/2/27 9:21	Microsoft Excel ...	6 KB
 circle-data-00148.csv	2024/2/27 9:21	Microsoft Excel ...	6 KB
 circle-data-00149.csv	2024/2/27 9:21	Microsoft Excel ...	6 KB
 circle-image-00000.png	2024/2/27 9:17	PNG 图片文件	141 KB
 circle-image-00001.png	2024/2/27 9:17	PNG 图片文件	142 KB
 circle-image-00002.png	2024/2/27 9:17	PNG 图片文件	145 KB
 circle-image-00003.png	2024/2/27 9:17	PNG 图片文件	142 KB
 circle-image-00004.png	2024/2/27 9:17	PNG 图片文件	142 KB
 circle-image-00005.png	2024/2/27 9:17	PNG 图片文件	144 KB

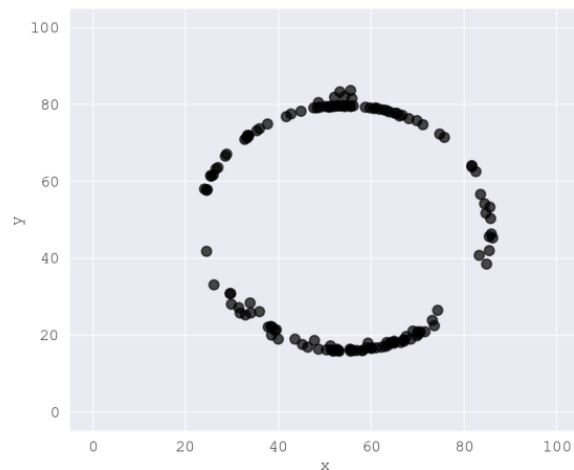
以下四个图形是分别间隔50个图像选取的，可以发现最开始的小恐龙嘴巴逐渐圆润，小手手也没子呜呜。第二张图还能看到它的丹鹤顶。**因此可以得到初步的观察结论：**

- 变化过程中主要改变集中在前期，前25%次扰动已经能产生明显的定向变化
- 变化过程后期几乎微动很小，大部分都在进行微调。**推测是因为许多点已经找到了局部最优解甚至是全局最优解，模拟退火算法再进行扰动很难或者不会再产生新解。**
- 我们可以注意到最后生成的"圆形"，偏横向的椭圆而不是标准的圆形。并且明显更多的点倾向于分布在上下两侧而不是左右两侧。**这很可能是由于初始的数据中可以很明显地发现小恐龙的纵向尤其是偏中间的数据点分布远大于横向数据点分布。也就是初始的数据分布不均衡导致的。这也是潜在的改进方向之一。**

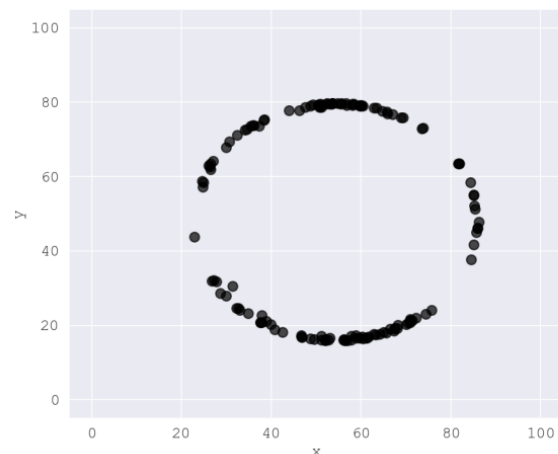




X Mean: 54.2691587
Y Mean: 47.8359580
X SD : 16.7623041
Y SD : 26.9341875
Corr. : -0.0601053



X Mean: 54.2616576
Y Mean: 47.8301060
X SD : 16.7669354
Y SD : 26.9382519
Corr. : -0.0605480



X Mean: 54.2650787
Y Mean: 47.8337227
X SD : 16.7667993
Y SD : 26.9336287
Corr. : -0.0627222

不过总体来说，实验较为成功。

任意形状探索

使用网址<http://robertgrantstats.co.uk/drawmydata.htm>生成任意形状的点阵数据集，让他经过有限次扰动，变为其他形状

修改代码

首先第一步为了后续将不同的数据和图片集能够储存在不同的位置，对程序中命令行添加参数，具体做法为修改以下几部分：

1. doc注释：

```

1  """
2  Usage:
3      same_stats.py run <shape_start> <shape_end> [<save_path>][<iters>]
4      [<decimals>][<frames>]
5      # ...blablabla...
6  """

```

2. main函数添加命令行参数:

```

1  # run <shape_start> <shape_end> [<save_path>][<iters>][<decimals>] 运行命
   令行
2  if __name__ == '__main__':
3      arguments = docopt(__doc__, version='Same Stats 1.0')
4      if arguments['run']:
5          save_path = None
6          it = 100000
7          de = 2
8          frames = 100
9          if arguments['<save_path>']:
10             save_path = arguments['<save_path>']
11             if arguments['<iters>']:
12                 it = int(arguments['<iters>'])
13             if arguments['<decimals>']:
14                 de = int(arguments['<decimals>'])
15             if arguments['<frames>']:
16                 frames = int(arguments['<frames>'])
17
18             shape_start = arguments['<shape_start>']
19             shape_end = arguments['<shape_end>']
20
21             if shape_start in initial_datasets and shape_end in all_targets:
22                 do_single_run(shape_start, shape_end, save_path=save_path,
23                               iterations=it, decimals=de, num_frames=frames)
24             else:
25                 print("***** One of those shapes isn't correct:")
26                 print("shape_start must be one of ", initial_datasets)
27                 print("shape_end must be one of ", all_targets)

```

3. 修改路径:

之后依次修改do_single_run, run_pattern, 最后在核心的趋近于指定形状的函数中, 添加路径部分:

```

1  # save this chart to the file 将此图表保存到文件中
2      for x in range(write_frames.count(i)):
3          if sv_path is not None:
4              directory = "results/" + sv_path + "/"
5              if not os.path.exists(directory):
6                  os.makedirs(directory)
7              save_scatter_and_results(r_good,
8                                      "results/" + sv_path + "/" +
9                                      target + "-image-" + format(int(frame_count),
10                                                                    '05'), 150,
11                                      labels=labels)

```

```

11         r_good.to_csv("results/" + sv_path + "/" + target + "-
data-" + format(int(frame_count), '05') + ".csv")
12     else:
13         save_scatter_and_results(r_good, "results/" + target +
"-image-" + format(int(frame_count), '05'), 150,
14                                     labels=labels)
15         r_good.to_csv("results/" + target + "-data-" +
format(int(frame_count), '05') + ".csv")
16         # save_scatter(r_good, target + "-image-
"+format(int(frame_count), '05'), 150)
17
18
19         frame_count = frame_count + 1
20     return r_good

```

使用给定网址

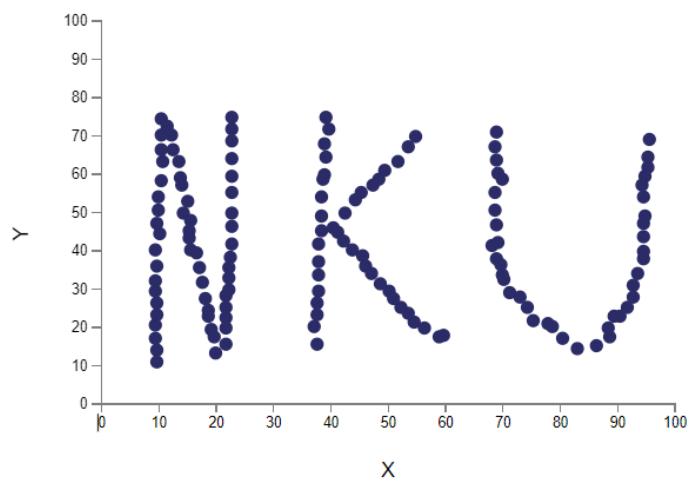
1. **生成数据：**使用上述网址生成如下图所示的名为**NKU**的数据点分布（hhh）。保存至csv后放到seed_datasets目录下。这里有个小坑也是，数据集列名是X和Y，但是程序里面只能识别x和y。

X name: X min: X max:

Y name: Y min: Y max:

N = 142 ; X mean = 44.0321 ; X SD = 28.5028 ; Y mean = 41.6873 ; Y SD = 17.6693 ; Pearson correlation = -0.0829

Slope = N/A ; Intercept = N/A



Reset chart

Save to CSV

filename:

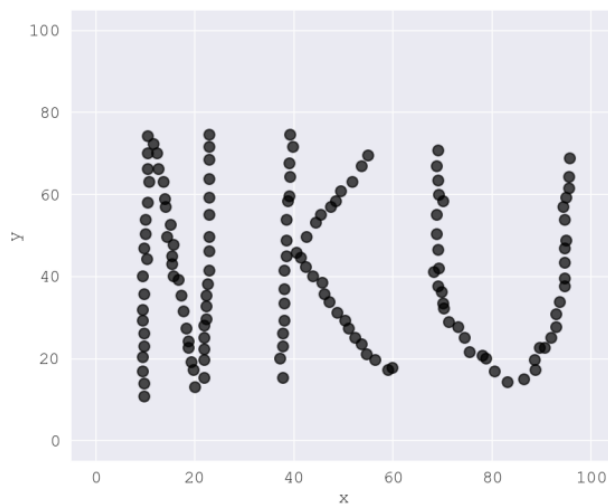
2. **程序运行：**添加新路径NKU后，再次运行，选择参数还是200000次迭代。

```
管理员: Anaconda Prompt (Anaconda)
(base) C:\Windows\System32>D:
(base) D:\>cd D:\Codes\BigData\Visualization
(base) D:\Codes\BigData\Visualization>python same_stats.py run mvdata circle NKU 200000 2 150
circle pattern: 100%|#####| 200001/200001 [03:29<00:00, 954.75it/s]
(base) D:\Codes\BigData\Visualization>
```

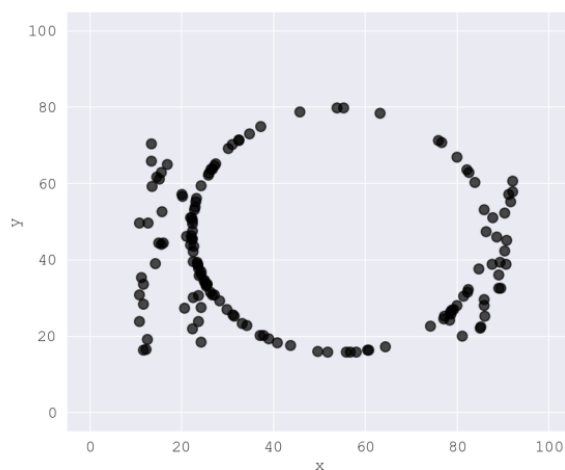
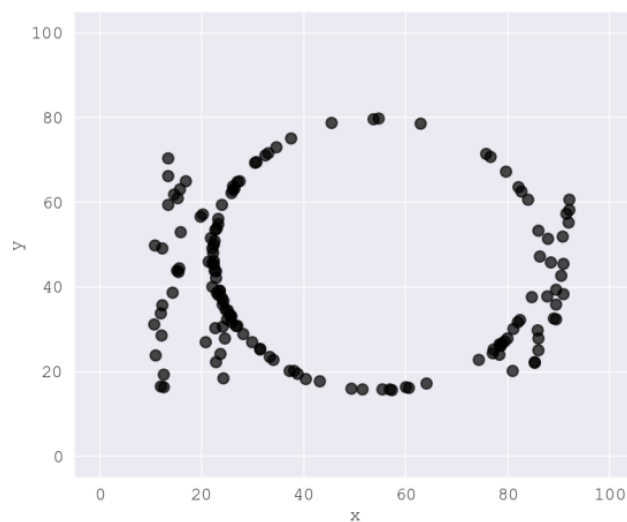
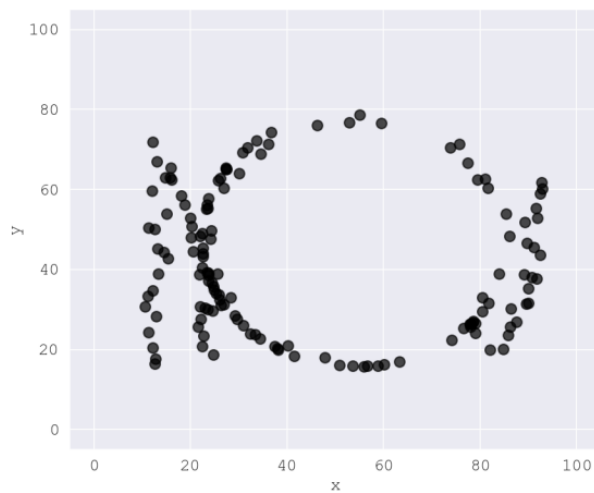
3. 复现结果:

相比于小恐龙，这次又出现了新的变化规律：

- 最重要的是经过相同的迭代轮数200000，其最后有相当多数量的离群点，尽管大部分点能够一定程度上勾勒出圆形的轮廓，但是目标形状完美程序明显不如恐龙。推测原因是点的分布更加不均匀，尤其是字母N和字母U在两侧聚集了大量的数据。使得分布不均衡。
- 局部最优解陷阱更为明显：模拟退火算法在搜索空间中随机游走，因此有时会陷入局部最优解，而无法跳出。这可能导致算法在全局最优解附近徘徊，而无法真正收敛。可以注意到同样也是最开始的25%数据点变化较大，后面那些离群点都没有很好地收敛，而是原地徘徊。这便是陷入了局部最优解陷阱，很难真正收敛。
- 时间性能较差：同样是140左右的数据，程序运行速度远低于小恐龙。



X Mean: 44.0323603
Y Mean: 41.6877628
X SD : 28.5025667
Y SD : 17.6694882
Corr. : -0.0829454



程序评价与探索

简单评价一下程序的性能，即扰动次数和目标形状完美程度的关系(尝试用可视化图分析哦)

性能评价

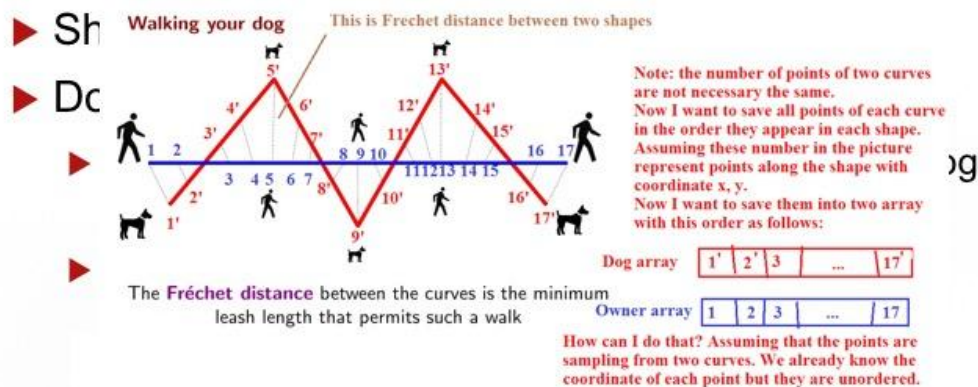
为了更好地评价模型的性能，这里以之前我使用的NKU数据集作为shape_start，circle作为shape_end，iterations即purturbations迭代次数从0开始，间隔为50000控制变量分别进行实验。对于目标形状完美程度的衡量，我们采用以下指标：

Fréchet距离和**Hausdorff距离**都是衡量两个形状之间相似度的方法。以下是这两种方法的简介以及如何在Python中实现它们：

1. **Fréchet距离**: Fréchet距离是一种衡量两个曲线相似性的度量, 它考虑了曲线上点的位置和顺序。可以使用 `frechetdist` 库来计算离散的Fréchet距离。

Frechet Distance

13



where, $\|C\| = \max_{k=1}^K \text{dist}(a_i^k, b_j^k)$

$D_F(L_i, L_j)$ is the Fréchet distance between trajectory segments L_i and L_j . Here, L_i and L_j are the trajectory segments whose lengths are m and n respectively. $K = \min(m, n)$. a_i^k and b_j^k are the k th points on trajectory segments L_i and L_j respectively. $dist(a_i^k, b_j^k)$ is the Euclidean distance between a_i^k and b_j^k .

H. A. It, "The computational geometry of comparing shapes," in *Efficient Algorithms*. Springer, 2009, pp. 235-248

Xie D, Li F, Phillips JM. Distributed Trajectory Similarity Search. *Pvldb*. 2017:1478-1489.

2. **Hausdorff距离**: Hausdorff距离是一种用来衡量两个点集之间的距离的度量，它考虑了点集中所有点的位置。在Python中，你可以使用 `scipy` 库中的 `directed_hausdorff` 函数来计算 Hausdorff距离。

Hausdorff distance

12

- Used to measure how far two trajectories are from each other

$$D_H(L_i, L_j) = \max(h(L_i, L_j), h(L_j, L_i))$$

where $h(L_i, L_j) = \max_{a \in L_i} (\min_{b \in L_j} (dist(a, b)))$

- [illegible]

H. A. It, "The computational geometry of comparing shapes," in *Efficient Algorithms*. Springer, 2009, pp. 235-248.

Xie D, Li F, Phillips JM. Distributed Trajectory Similarity Search. *Pvldb*. 2017:1478-1489.

评价函数

为了更好地评价模型的性能，我额外在utils文件夹中写了一个评价脚本evaluation.py，来更方便地一键式对程序进行模型评价。

程序仿照same_stats.py采用命令行的模型评价方式来运行。主体分为以下函数

1. get_points_on_circle

```
1 def get_points_on_circle(cx, cy, r, num_points):
2     """
3     This function will generate points on a circle based on the hard-
4     coded center and radius in the paper.
5     :param cx: center x of the circle
6     :param cy: center y of the circle
7     :param r: radius of the circle
8     :param num_points: points of circle to be generated and sampled, it
9     should be identical to the number of the
10    approximated trajectory to guarantee a same dimension for distance
11    calculation.
12    :return:
13    """
14    circle_points = []
15    for i in range(num_points):
16        angle = math.pi * 2 * i / num_points
17        x = cx + r * math.cos(angle)
18        y = cy + r * math.sin(angle)
19        circle_points.append((x, y))
20    return circle_points
```

2. do_evaluation

```
1 def do_evaluation(file_path):
2     """
3     This function will calculate the Frechet distance and Hausdorff
4     distance
5     to give a rough evaluation of how approximated the model can be.
6     :param file_path: given parameter of the file path, prepared to be
7     loaded
8     :return:
9     """
10    circle_points = get_points_on_circle(54.26, 47.83, 30, 142)
11    target_trajectory = circle_points
12    frechet_dists = []
13    hausdorff_dists = []
14    for i in range(7):
15        csv_path = file_path + "/circle-data-0000" + str(i) + ".csv"
16        approximated_trajectory = pandas.read_csv(csv_path, usecols=[1,
17        2]).values.tolist()
18        approximated_trajectory = np.array(approximated_trajectory)
19        target_trajectory = np.array(target_trajectory)
20        print("-----Evaluation-----")
21        print("Iteration: ", 50000 * i)
22        # print("Target trajectory: ", target_trajectory)
23        # print("Approximated trajectory: ", approximated_trajectory)
24        print("-----Frechet Distance-----")
```

```

22     frechet_dist = frdist(target_trajectory,
approximated_trajectory)
23     print("The Frechet distance is:" + str(frechet_dist))
24     frechet_dists.append(frechet_dist)
25     print("-----Hausdorff Distance-----")
26     hausdorff_dist = directed_hausdorff(target_trajectory,
approximated_trajectory)[0]
27     print("The Hausdorff distance is:" + str(hausdorff_dist))
28     hausdorff_dists.append(hausdorff_dist)
29     # Call the plot function at the end of do_evaluation
30     plot_evaluation(frechet_dists, hausdorff_dists)

```

3. plot_evaluation:进行可视化图分析。

```

1  def plot_evaluation(frechet_dists, hausdorff_dists):
2      """
3          This function will plot the evaluation result of the approximated
trajectory
4          Based on the Frechet distance and Hausdorff distance.
5          :param frechet_dists: Calculated Frechet distance from do_evaluation
6          :param hausdorff_dists: Calculated Hausdorff distance from
do_evaluation
7          :return:
8          """
9          iterations = [i * 50000 for i in range(7)]
10         plt.figure(figsize=(10, 5))
11         plt.plot(iterations, frechet_dists, marker='o', color='b',
label='Frechet Distance')
12         plt.xlabel('Iterations')
13         plt.ylabel('Frechet Distance')
14         plt.title('Frechet Distance of Approximated Trajectory')
15         plt.xticks(iterations)
16         plt.grid(True)
17         plt.legend()
18         plt.show()
19
20         plt.figure(figsize=(10, 5))
21         plt.plot(iterations, hausdorff_dists, marker='v', color='r',
label='Hausdorff Distance')
22         plt.xlabel('Iterations')
23         plt.ylabel('Hausdorff Distance')
24         plt.title('Hausdorff Distance of Approximated Trajectory')
25         plt.xticks(iterations)
26         plt.grid(True)
27         plt.legend()
28         plt.show()

```

4. main

```

1  """
2  Usage:
3      evaluation.py run [<relative_path>]
4      evaluation.py -h | --help
5
6      Based on the code in same_stats.py,

```

```

7      Same Stats, Different Graphs Generating Datasets with Varied
      Appearance and Identical Statistics through Simulated Annealing
8      This is an evaluation script for model's performance on
9      approximating the target trajectory.
10     The script will calculate the Frechet distance and Hausdorff
      distance
11     between the target trajectory and the approximated trajectory.
12
13     Version1.0
14     Now the evaluation script is only based on given hard-coded target
      trajectory such as a circle.
15     And it has been extended to be cover any interval between sampling
      data during the transformation in the model.
16     To be continue.....
17
18     """
19     import numpy as np
20     import pandas
21     from docopt import docopt
22     from frechetdist import frdist
23     from scipy.spatial.distance import directed_hausdorff
24     import math
25     from matplotlib import pyplot as plt
26
27     if __name__ == '__main__':
28         arguments = docopt(__doc__, version='Evaluation 1.0')
29         file_path = None
30         if arguments['run']:
31             if arguments['<relative_path>']:
32                 file_path = "../results/" + arguments['<relative_path>']
33                 do_evaluation(file_path)
34             else:
35                 print("No relative path")

```

评价结果

采用命令行运行对之前采集到的分为7段的变化过程中的数据的评价。即NKU数据集作为shape_start, circle作为shape_end, iterations即perturbations迭代次数从0开始, 间隔为50000控制变量分别进行实验。结果如下:

管理员: Anaconda Prompt (Anaconda)

```
(base) C:\Windows\System32>D:
```

```
(base) D:\>cd D:\Codes\BigData\Visualization\utils
```

```
(base) D:\Codes\BigData\Visualization\utils>python evaluation.py run NKU
```

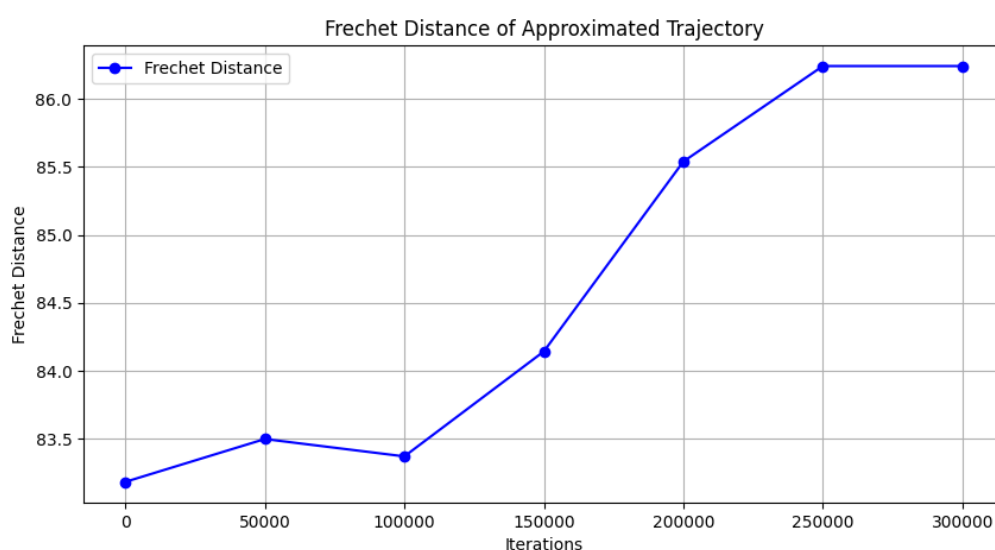
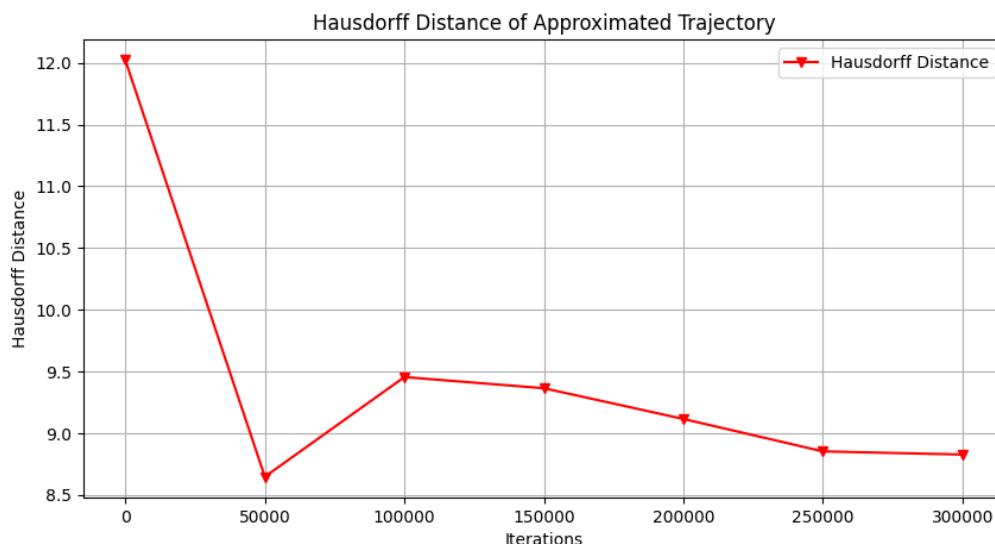
```

-----Evaluation-----
Iteration:  0
-----Frechet Distance-----
The Frechet distance is:83.18524709706642
-----Hausdorff Distance-----
The Hausdorff distance is:12.021022474485576
-----Evaluation-----
Iteration:  50000
-----Frechet Distance-----
The Frechet distance is:83.50017722993043
-----Hausdorff Distance-----
The Hausdorff distance is:8.648444611247777
-----Evaluation-----
Iteration:  100000
-----Frechet Distance-----
The Frechet distance is:83.37248839890955
-----Hausdorff Distance-----
The Hausdorff distance is:9.455344977201074
-----Evaluation-----
Iteration:  150000
-----Frechet Distance-----
The Frechet distance is:84.14410458877674
-----Hausdorff Distance-----
The Hausdorff distance is:9.365086090087509
-----Evaluation-----
Iteration:  200000
-----Frechet Distance-----
The Frechet distance is:85.53956992655264
-----Hausdorff Distance-----
The Hausdorff distance is:9.116265203098596
-----Evaluation-----
Iteration:  250000
-----Frechet Distance-----
The Frechet distance is:86.24157798779954
-----Hausdorff Distance-----
The Hausdorff distance is:8.853568833005781
-----Evaluation-----
Iteration:  300000
-----Frechet Distance-----
The Frechet distance is:86.24157798779954
-----Hausdorff Distance-----
The Hausdorff distance is:8.827914409458618

(base) D:\Codes\BigData\Visualization\utils>

```

shape_start	shape_end	iterations(purturbations)	Hausdorff	Fréchet
NKU	circle	1	12.021022474485576	83.18524709706642
NKU	circle	50000	8.648444611247777	83.50017722993043
NKU	circle	100000	9.455344977201074	83.37248839890955
NKU	circle	150000	9.365086090087509	84.14410458877674
NKU	circle	200000	9.116265203098596	85.53956992655264
NKU	circle	250000	8.853568833005781	86.24157798779954
NKU	circle	300000	8.827914409458618	86.24157798779954



由可视化和图表化结果可以发现：

- 扰动次数和目标形状完美程度的关系并不一定呈现正相关，可以看到呈现一个几乎波动的状态，这是由于扰动导致不断寻找局部最优解的特征。
- 由Hausdorff距离能很清楚地体现只有最开始迭代的50000次与目标形状的差异大幅变化，之后迭代情况下都波动较小，在慢慢趋近，这也与我们之前用点状图直观地感受结果相同。

优化方向

1. **Target形状方向的硬编码**：正如作者的doc注释所言，程序的目标形状方向是硬编码的，一定程度上限制了目标形状的灵活性和泛性。
 - **改进的方向**包括令目标形状数据也可以像mydata那样指定或随机生成。
2. **扰动函数的性能**：因为扰动函数是基于模拟退火算法，所以可以先从该算法自带的问题角度进行分析。

模拟退火算法是一种用于优化问题的启发式算法，其灵感来源于固体退火过程。尽管它在许多情况下表现良好，但也存在一些不足之处。

- **局部最优解陷阱**：模拟退火算法在搜索空间中随机游走，因此有时会陷入局部最优解，而无法跳出。这可能导致算法在全局最优解附近徘徊，而无法真正收敛。
- **收敛速度慢**：模拟退火算法的收敛速度相对较慢。尤其是在高维问题中，需要大量的迭代才能达到最优解。

- **参数选择敏感**: 模拟退火算法的性能受到参数设置的影响。选择合适的初始温度、退火速率等参数并不容易, 需要经验或试错。

由此结合具体的算法步骤代码, 我提出几点暂时性的优化建议:

```
1  #
2  # This is the function which does one round of perturbation
3  # df: is the current dataset
4  # initial: is the original dataset
5  # target: is the name of the target shape
6  # shake: the maximum amount of movement in each iteration
7  # 这是一个进行了一轮扰动的函数
8  # df:当前数据集
9  # initial:原始数据集
10 # target:目标形状的名称
11 # shake:每次迭代的最大移动量
12 #
13 def perturb(df, initial, target='circle',
14             line_error=1.5,
15             shake=0.1,
16             allowed_dist=2,
17             temp=0,
18             x_bounds=[0, 100], y_bounds=[0, 100],
19             custom_points=None):
20     # take one row at random, and move one of the points a bit
21     # 随机取一行, 并移动一个点
22     row = np.random.randint(0, len(df))
23     i_xm = df['x'][row]
24     i_ym = df['y'][row]
25
26     # this is the simulated annealing step, if "do_bad", then we are
    willing to
27     # accept a new state which is worse than the current one
28     # 这是模拟退火步骤, 如果“do_bad”, 那么我们愿意接受比当前状态更糟糕的新状态
29     do_bad = np.random.random_sample() < temp
30
31     while True:
32         xm = i_xm + np.random.randn() * shake
33         ym = i_ym + np.random.randn() * shake
34
35         if target == 'circle':
36             # info for the circle 圆形点阵中x, y坐标的均值和方差
37             cx = 54.26 # 均值
38             cy = 47.83 # 均值
39             r = 30
40
41             dc1 = dist([df['x'][row], df['y'][row]], [cx, cy])
42             dc2 = dist([xm, ym], [cx, cy])
43
44             old_dist = abs(dc1 - r)
45             new_dist = abs(dc2 - r)
46
47         elif target == 'bullseye':
48             # info for the bullseye 双环形点阵中x, y坐标的均值和方差
49             cx = 54.26
50             cy = 47.83
51             rs = [18, 37]
52
```

```

53         dc1 = dist([df['x'][row], df['y'][row]], [cx, cy])
54         dc2 = dist([xm, ym], [cx, cy])
55
56         old_dist = np.min([abs(dc1 - r) for r in rs])
57         new_dist = np.min([abs(dc2 - r) for r in rs])
58
59         elif target == 'dots':
60             # create a grid of "cluster points" and move if you are
61             # getting closer
62             # (or are already close enough)
63             # 创建一个“群集点”网格，如果你越来越后就移动
64             # (或者已经足够接近了)
65             xs = [25, 50, 75]
66             ys = [20, 50, 80]
67
68             old_dist = np.min([dist([x, y], [df['x'][row], df['y']
69             [row]]) for x, y in itertools.product(xs, ys)])
70             new_dist = np.min([dist([x, y], [xm, ym]) for x, y in
71             itertools.product(xs, ys)])
72
73         elif target in line_shapes:
74             lines = get_points_for_shape(target)
75
76             # calculate how far the point is from the closest one of
77             # these
78             # 计算这个点离最近的一个点有多远
79             old_dist = np.min([DistancePointLine(i_xm, i_ym, l[0][0],
80             l[0][1], l[1][0], l[1][1]) for l in lines])
81             new_dist = np.min([DistancePointLine(xm, ym, l[0][0], l[0]
82             [1], l[1][0], l[1][1]) for l in lines])
83
84             # check if the new distance is closer than the old distance
85             # or, if it is less than our allowed distance
86             # or, if we are do_bad, that means we are accepting it no matter
87             # what
88             # if one of these conditions are met, jump out of the loop
89             # 检查新距离是否比旧距离更近
90             # 或，它是否小于我们允许的距离
91             # 或，是否运行了do_bad，那就意味着我们无论如何都要接受
92             # 如果满足其中一个条件，就跳出循环
93
94             if ((new_dist < old_dist or new_dist < allowed_dist or do_bad)
95             and
96             y_bounds[0] < ym < y_bounds[1] and x_bounds[0] < xm <
97             x_bounds[1]):
98                 break
99
100             # set the new data point, and return the set
101             # 设置新的数据点，并返回集合
102             df['x'][row] = xm
103             df['y'][row] = ym
104             return df

```

改进建议（个人看法）包含以下几点：

- **更灵活的退火策略**：由代码（或者下面提取的论文中的伪代码）可以看到这里所采用的退火算法是最基础的退火算法。这就可能带有退火算法的**局部最优解陷阱**。为了克服这一问题，可以考虑引入**更灵活的退火策略，例如自适应退火速率或动态调整温度**。


```

1: current_ds ← initial_ds
2: for  $x$  iterations, do:
3:   test_ds ← PERTURB(current_ds, temp)
4:   if ISERROROK(test_ds, initial_ds):
5:     current_ds ← test_ds
6:
7: function PERTURB( $ds$ ,  $temp$ ):
8:   loop:
9:     test ← MOVERANDOMPOINTS( $ds$ )
10:    if FIT( $test$ ) > FIT( $ds$ ) or  $temp$  > RANDOM():
11:      return test

```

- **定向变化的中心性：**可以发现圆形等形状的均值和方差都已经是硬编码的，这可能导致数据点向某个中心整体倾向，比较耗费迭代次数。如果直接选择各个数据点的几何中心自适应地生成，可以更好地理论上更快地产生指定形状，而不是在数据点分布较为分散时候整体向指定中心偏移变化。
- **时间复杂度优化：**算法部分时间复杂度较高。如每个迭代重复的判断，都可以进一步优化。

留在最后

作为课程的第一项任务Task1，复现论文这个让我留下了很深的印象。作为计算机和网络空间安全学院的学生，大数据和相关领域一直是我很感兴趣的方向。本次我也找到了很多可以改进的方向，后续如果时间允许，我会将我的改进结果在未来进一步与老师讨论，希望老师能给出一些建议，主要是希望真的能发论文哈哈。