粒子群算法实验————解决TSP问题

> 本报告使用粒子群算法解决TSP问题

1. 实验背景

旅行商问题，即TSP问题（Traveling Salesman Problem），也叫旅行推销员问题、货郎担问题，是数学领域中著名问题之一。假设有一个旅行商人要拜访n个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

旅行推销员问题是图论中最著名的问题之一，即“已给一个n个点的完全图，每条边都有一个长度，求总长度最短的经过每个顶点正好一次的封闭回路”。Edmonds，Cook和Karp等人发现，这批难题有一个值得注意的性质，对其中一个问题存在有效算法时，每个问题都会有有效算法。

迄今为止，这类问题中没有一个找到有效算法。倾向于接受NP完全问题（NP-Complete或NPC）和NP难题（NP-Hard或NPH）不存在有效算法这一猜想，认为这类问题的大型实例不能用精确算法求解，必须寻求这类问题的有效的近似算法。

2. 粒子群算法

2.1 算法简介

粒子群算法，也称粒子群优化算法或鸟群觅食算法（Particle Swarm Optimization），缩写为 PSO。PSO 算法属于进化算法的一种，和模拟退火算法相似，它也是从随机解出发，通过迭代寻找最优解，它也是通过适应度来评价解的品质，但它比遗传算法规则更为简单，它没有遗传算法的“交叉”(Crossover) 和“变异”(Mutation) 操作，它通过追随当前搜索到的最优值来寻找全局最优。这种算法以其实现容易、精度高、收敛快等优点引起了学术界的重视，并且在解决实际问题中展示了其优越性。粒子群算法是一种并行算法。

PSO模拟鸟群的捕食行为。设想这样一个场景：一群鸟在随机搜索食物。在这个区域里只有一块食物。所有的鸟都不知道食物在那里。但是他们知道当前的位置离食物还有多远。那么找到食物的最优策略是什么呢。最简单有效的就是搜寻目前离食物最近的鸟的周围区域。

PSO从这种模型中得到启示并用于解决优化问题。PSO中，每个优化问题的解都是搜索空间中的一只鸟。我们称之为“粒子”。所有的粒子都有一个由被优化的函数决定的适应值(fitness value)，每个粒子还有一个速度决定他们飞翔的方向和距离。然后粒子们就追随当前的最优粒子在解空间中搜索。

PSO 初始化为一群随机粒子(随机解)。然后通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个"极值"来更新自己。第一个就是粒子本身所找到的最优解，这个解叫做个体极值pBest。另一个极值是整个种群目前找到的最优解，这个极值是全局极值gBest。另外也可以不用整个种群而只是用其中一部分作为粒子的邻居，那么在所有邻居中的极值就是局部极值。

2.2 算法结合TSP

由于TSP是离散问题,所以普通的PSO方法无法使用,必须使用广义的PSO算法.

广义粒子群算法模型和遗传算法相当类似，目前网上有关于粒子群算法求解TSP的很多论文或代码都是基于广义粒子群算法的，说简单点就是进化思想，用交叉变异代替了基本粒子群算法的迭代公式，当然他们也还是有粒子

群优化的本质思想的，如与全局最优编码交叉，与局部最优编码交叉，变异等都是源自于基本粒子群算法的迭代公式。

在这样的背景下,我们设计了独特的速度更新公式,利用概率来作为参数调整的方式,使得原有的公式可以继续扩展的使用.

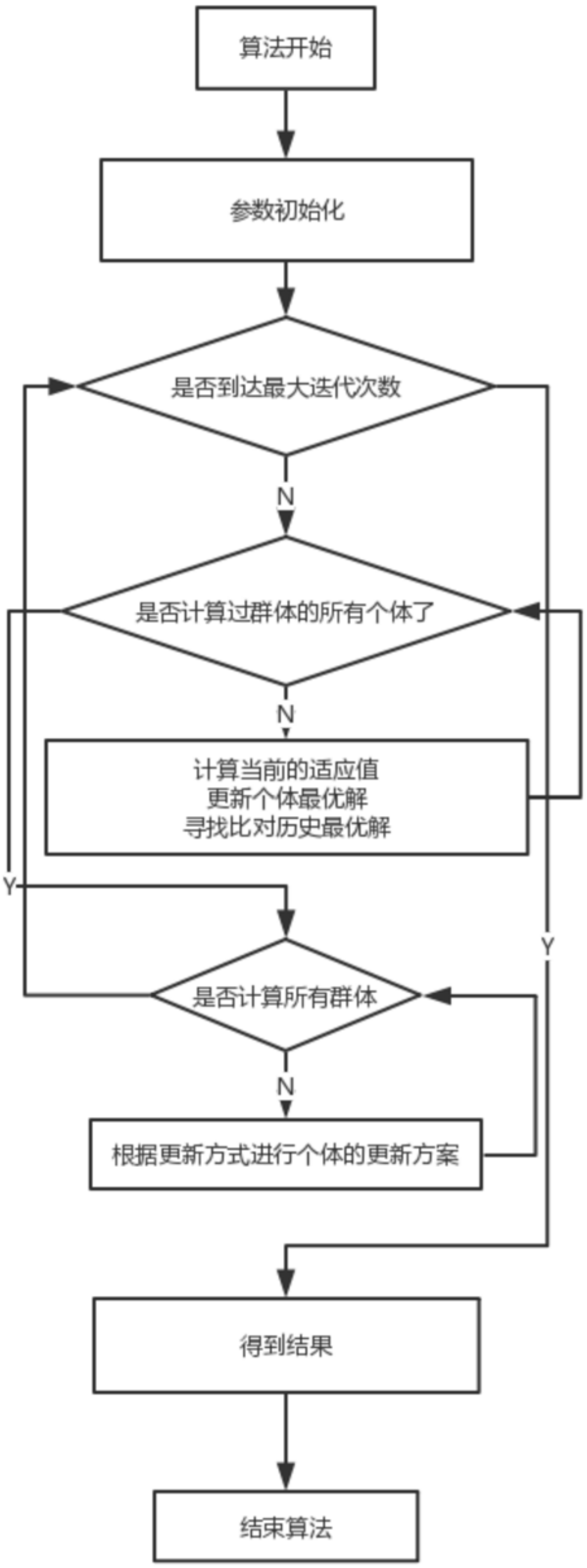
具体算法详见设计细节

3. 算法设计流程

3.1 算法流程图

如下图所示:

3.2 算法实现过程



使用python3作为设计语言，其中本次使用的版本是python3.6，其中使用了常规的的库有：

1. time
2. copy
3. random
4. math

使用的非内置库有：

1. numpy
2. matplotlib

3.2.1 算法公式

> PSO 算法包含的公式有：
>
> 1. 速度更新公式
> 2. 位置更新公式
>   由于是在离散场景下的实际应用问题，所以这里还需要改变部分以适应TSP问题的要求

1. 速度更新公式

$$V_{id}^{k+1}=V_{id}^{k}+c_1*\eta*(p_{id}^{k}-x_{id}^{k})+c_2*\xi*(p_{gd}^{k}-x_{id}^{k})$$

- 下标id 表示群体id
- k 表示的是迭代次数
- c1,c2是学习因子
- η,$\xi\in[0,1]$
- p_{id}表示本个体的历史最优
- p_{gd}表示本群体的最优

2. 位置更新公式

$$x_{id}^{k+1}=x_{id}^{k}+v_{id}^{k+1}$$

   利用加操作，这样的操作就可以
3. 与TSP问题协调
   以上公式为PSO的经典计算公式，但是在结合TSP问题中，同时出现了如下问题：

- 编码要求中无法使用浮点数为TSP问题编码
- TSP问题无法进行直接速度相加

为了解决以上的问题，在处理过程中使用了一定的变通的手法：

> 粒子群本质上是使用历史最好和群体最好的结果来不断优化群体，而其速度就是定义为更新某个群体的方式
> 在顺序编码中最好的更新方式就是对于数据位置的交换，因在，在次数的更新速度我们定义为以一定的概率接受某两个位置的交换。
> 也就是说，以概率的方式解释上述的速度更新公式和位置更新公式。

具体的处理方式和代码参见下一节

3.2.2 算法设计细节

1. TSP处理
   首先需要定义图本身
   首先定义一个点类

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

随后定义图类，注意到其中的 `ask_distance_for_plan()` 方法，这样就可以直接用这个图对于一个可能的序列获取改方案的结果.

```
class Graph:
    def __init__(self, n):
        self.points = []
        self.point_n = n

    def add_point(self, p):
        self.points.append(p)
        self.point_n += 1

    def ask_distance(self, i, j):
        point_x = self.points[i]
        point_y = self.points[j]
        return math.sqrt(sqr(point_x.x - point_y.x) + sqr(point_x.y - point_y.y))

    def ask_distance_for_plan(self, plan):
        res = 0.0
        for i in range(1, self.point_n):
            res += self.ask_distance(plan[i], plan[i - 1])
        res += self.ask_distance(plan[0], plan[self.point_n - 1])
        # print(plan,' ',res)
        return res

    def show(self):
        for item in self.points:
            print(item.x, " ", item.y)
```

2. 定义Swarm类
   下面定义粒子群类,需要的属性是:

- 当前的方案
- 当前的方案结构
- 历史最优方案
- 历史最优方案的结果

```
class Swarm:
    def __init__(self, plan):
        self.plan = copy.copy(plan)
        self.fitness = 0.0
        self.best = 0.0
        self.best_plan = []
```

3. 定义PSO类
   下面定义PSO类
   为了保证代码设计的完整性,所以这里的解释在注释中:

```python
class PSO:
    """
        构造函数,实际上设置基础参数
        max_group 表示群体个数
        max_time 表示最大迭代次数
        c1,c2表示学习银子
    """
    def __init__(self, max_group=100, max_time=1000, c1=1, c2=1):
        self.MAX_TIME = max_time
        self.MAX_GROUP = max_group
        self.c1 = c1
        self.c2 = c2

    """
        给定向量的大小,返回一个不重复的随机向量
    """
    def init_plan(self, n):
        # 设置随机数种子
        random.seed(time.time())
        init_plan = []
        for i in range(0, n):
            init_plan.append(i)

        for i in range(n):
            # 随机两个不相同的数字
            j = random.randint(0, n - 1)
            while i == j:
                j = random.randint(0, n - 1)
            init_plan[i], init_plan[j] = init_plan[j], init_plan[i]

        return init_plan

    """
        给定群体大小,返回所获得的拥有不相同的数据的群体
    """
    def init_group(self, n, graph):
        group = []
        for i in range(self.MAX_GROUP):
            new_plan = self.init_plan(n)

            while new_plan in group:
                new_plan = self.init_plan(n)
            group.append(new_plan)

        return group

    """
        实际的运行函数
        传入的数据是:
            * 群体数据
            * 群体图本身
    """
    def run(self, n, graph):
        # 设置随机数种子
        random.seed(time.time())
        group = self.init_group(n, graph)

        # 获取初始群体
        swarms = []
        for item in group:
```

```python
        tmp = Swarm(item)
        tmp.fitness = graph.ask_distance_for_plan(item)
        tmp.best = graph.ask_distance_for_plan(item)
        tmp.best_plan = copy.copy(tmp.plan)
        tmp.speed = self.init_plan(n)
        swarms.append(tmp)

    i_time = 0

    # 最优值记录
    swarm_best = {
        "fitness": swarms[0].fitness,
        "plan": copy.copy(swarms[0].plan)
    }

    while i_time < self.MAX_TIME:

        # 首先对于所有的个体进行更新
        for bird in swarms:
            bird.fitness = graph.ask_distance_for_plan(bird.plan)
            if bird.fitness < bird.best:
                bird.best_plan = copy.copy(bird.plan)
                bird.best = bird.fitness

            if bird.best < swarm_best["fitness"]:
                swarm_best["fitness"] = bird.best
                swarm_best["plan"] = copy.copy(bird.plan)

        print(swarm_best)

        # 注意这个就是速度更新
        # 存储的是交换的位置操作以满足不重复性
        # 同时通过概率的方式方式保存前面的接受常数的概念
        temp_speed = []

        # 更新每个粒子
        for bird in swarms:
            temp_speed.clear()

            # print(bird.plan)
            # 设置eta,xi
            # 这个地方也可以设置乘全局的,也就是粒子群本身的属性中
            eta = 0.9
            xi = 0.85

            for i in range(n):
                if bird.plan[i] != swarm_best["plan"][i]:
                    # 存储的是
                    #   (i,j,p)
                    #   i,j,分别表示要交换的位置
                    #   p表示接受的概率
                    swap_operator = (i, swarm_best["plan"].index(bird.plan[i]), eta)
                    temp_speed.append(swap_operator)
                    u = swap_operator[0]
                    v = swap_operator[1]
                    bird.plan[u], bird.plan[v] = bird.plan[v], bird.plan[u]

            for i in range(n):
                if bird.plan[i] != bird.best_plan[i]:
                    swap_operator = (i, bird.best_plan.index(bird.plan[i]), xi)
                    temp_speed.append(swap_operator)
```

```
                u = swap_operator[0]
                v = swap_operator[1]
                bird.plan[u], bird.plan[v] = bird.plan[v], bird.plan[u]

        # 下面是用一个随机的变量来判断是否接受交换
        for item in temp_speed:
            rate = random.random()
            if rate <= item[2]:
                u = item[0]
                v = item[1]
                bird.plan[u], bird.plan[v] = bird.plan[v], bird.plan[u]

        # 为了保证不会落入到历史最优解中不出来,会给出变异操作,以构造出其他解
        if bird.plan == swarm_best["plan"]:
            u = random.randint(0, n - 1)
            v = random.randint(0, n - 1)
            while v == u:
                v = random.randint(0, n - 1)
            bird.plan[u], bird.plan[v] = bird.plan[v], bird.plan[u]

    i_time += 1
return swarm_best
```

4. 实验结果

4.1 实验数据

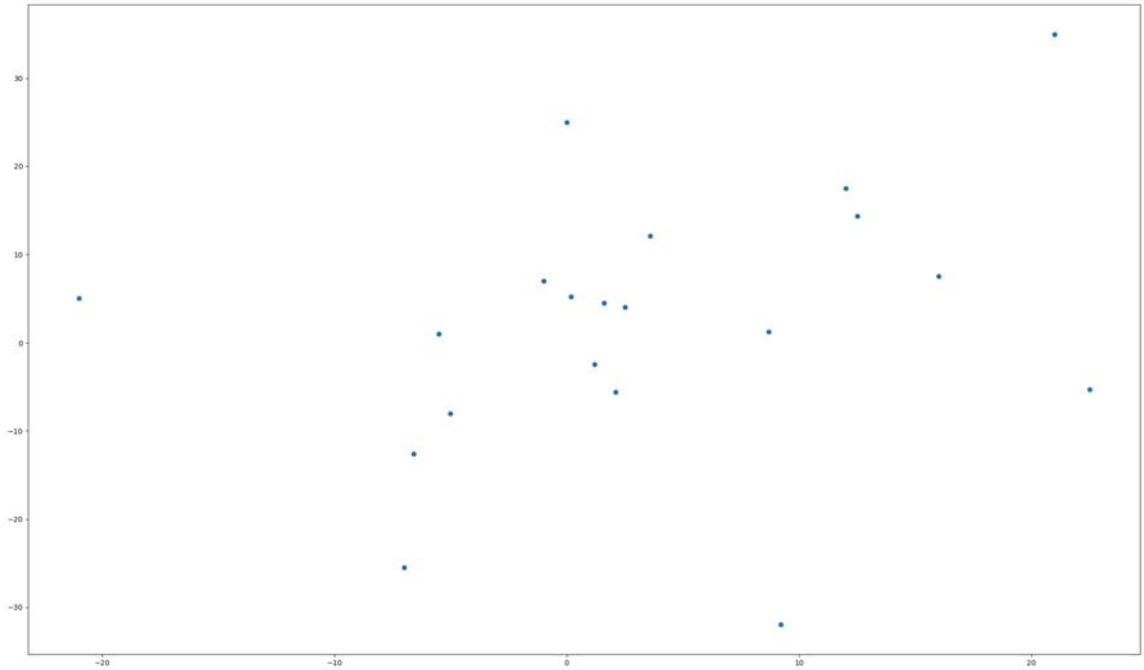
使用的实验数据如下:
给出所有点的坐标

```
A 2.5 4.0
B 1.2 -2.4
C 8.7 1.2
D 3.6 12.1
E -5.5 0.94
F -6.6 -12.6
G 0.18 5.219
H 12.5 14.3609
I 22.5 -5.26
J 1.61 4.5
K 2.1 -5.6
L 0 25
M 9.2 -32
N -1 7
O -5 -8
P 21 35
Q 16 7.5
R -21 5
S -7 -25.5
T 12 17.5
```

给出读入数据的python代码:

```python
graph = Graph(0)
with open("in.txt","r") as f:
    lines = f.readlines()
    for line in lines:
        line = str(line)
        # print(line)
        items = line.split(' ')
        x = float(items[1])
        y = float(items[2])
        # print('x =',x,' y = ',y)
        graph.add_point(Point(x,y))
```

给出一个简单的图:

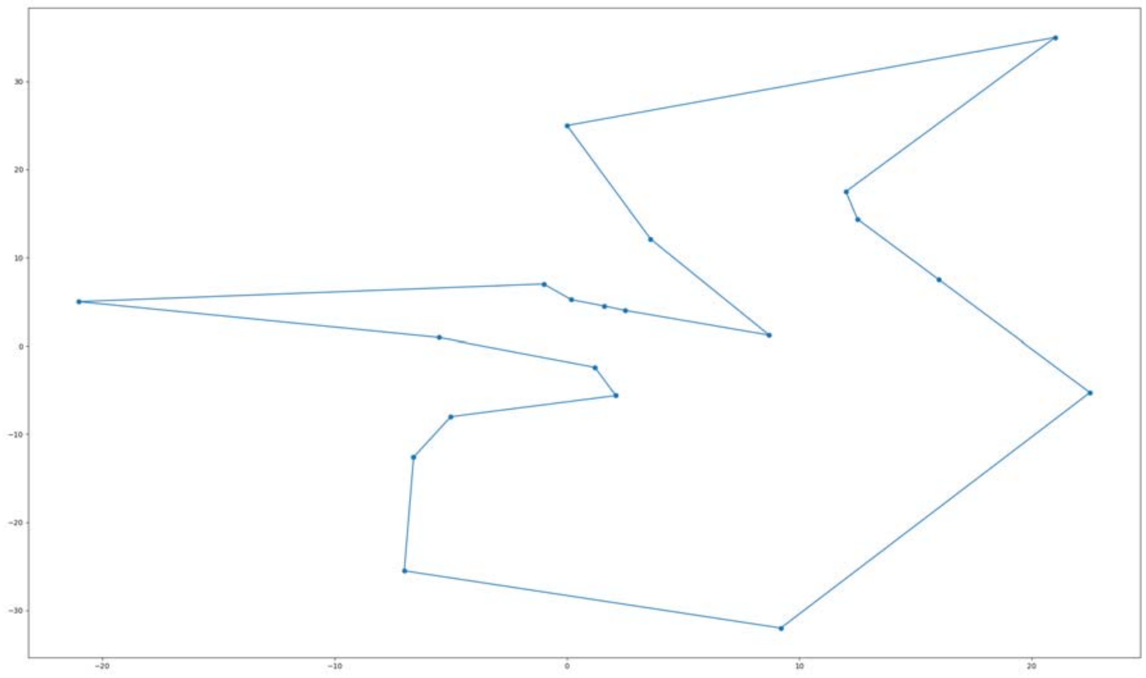


4.2 实验结果展示

最终运行的结果:

```
{'fitness': 224.6515394977227, 'plan': [5, 14, 10, 1, 4, 17, 13, 6, 9, 0, 2, 3, 11, 15, 19, 7, 16, 8, 12,
```

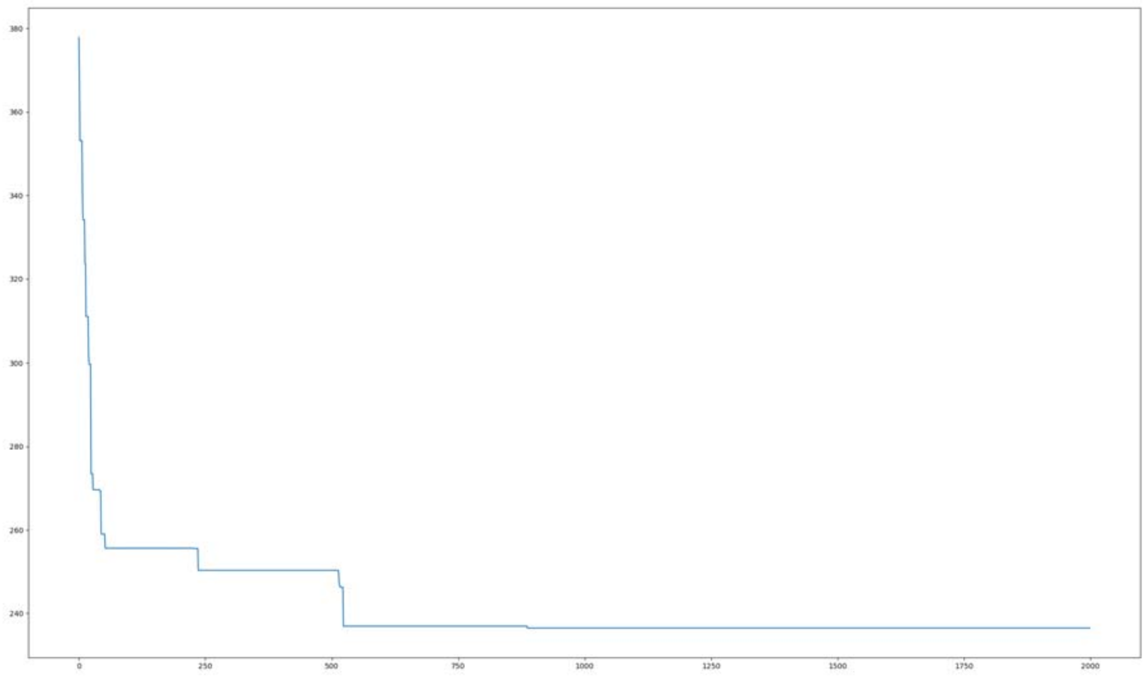
也就是有图:



4.2 实验结果分析

4.2.1迭代次数对于最终结果的影响

记录生成过程中的结果:



可以看到的是,粒子群在一开始的下降中是非常迅速的.所以在迭代次数相同的状态下,往往粒子群算法会有一个不错的结果.

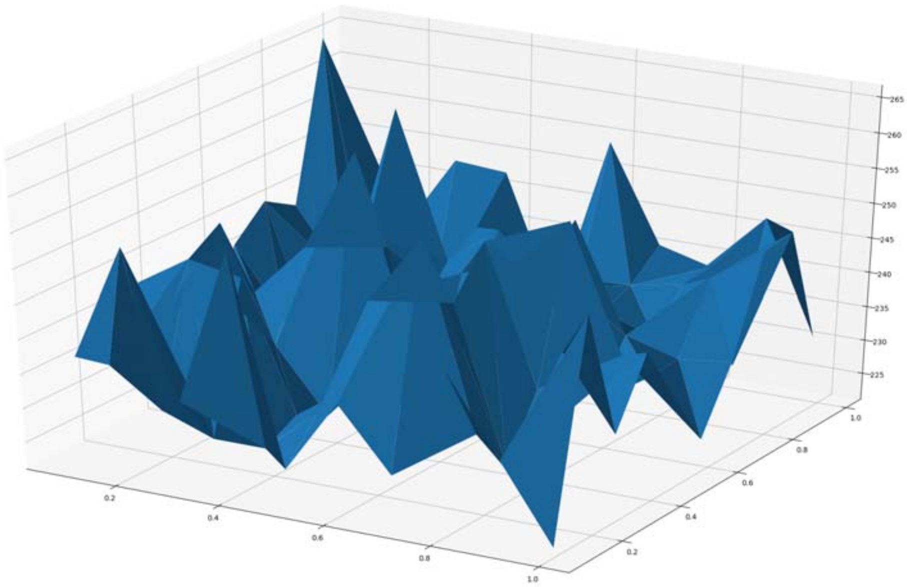
4.2.2 改变接受率对于最终结果的影响

在原来的例子中,我们的的接受率取的是固定的$\eta = 0.85, \xi = 0.9$,我们改变这个值的组合:
我们分别探究其组合对于其数值的影响.

我们计算如下的集合的结果的组合并绘制三维图像:

```
eta = [0.09, 0.19, 0.29, 0.39, 0.49, 0.59, 0.69, 0.79, 0.89, 0.99]
xi = [0.09, 0.19, 0.29, 0.39, 0.49, 0.59, 0.69, 0.79, 0.89, 0.99]
```

最终的结果为:



可以看出在向群体最优解方向的更新受限制的时候,结果会出现明显变差的结果,也就是说向群体最优解靠近比向自己的结果靠近在TSP问题中显得更加重要,这也说明了PSO的优越性在于了这两种更新方式的综合