# 3.3 更新策略

## 3.3.1 1维度权重决策、复杂网络更新和2维度布局优化——空间决策支持系统

# IPython extension to reload modules before executing user code.  
%load\_ext autoreload   
# Reload all modules (except those excluded by %aimport) every time before executing the Python code typed.  
%autoreload 2   
from usda import datasets as usda\_datasets  
from usda import data\_visualization as usda\_vis  
from usda import pattern\_signature as usda\_signature  
from usda import utils as usda\_utils  
from usda import meta\_heuristics as usda\_heuristic  
from usda import network as usda\_network  
  
import mapclassify  
import numpy as np  
import cc3d  
import copy  
import matplotlib.pyplot as plt  
import random  
import itertools  
import networkx as nx  
from networkx.algorithms import approximation as approx  
import math  
from toolz import partition  
from collections import defaultdict  
from scipy.signal import convolve2d  
  
np.set\_printoptions(linewidth=np.inf)

### 3.3.1.1 1维度权重决策

在“权重决策”一章，对于 BWM 和 IDOCRIW方法分别采用了GWO 和 GA 元启发式算法求解方程组。元启发式算法（一种新型的优化算法）通常用于存在许多可能解决方案（多备选方案）的多变量（多准则）问题中，并且结果会根据这些变量的组合不同发生较大的变化。优化算法通过评分（由约束条件（constraints）给出）许多不同的解决方案确定它们的质量（对评分结果进行排序）来找到问题的最佳答案（题解）。优化算法通常用于由于解决的方案太多而无法全部遍历尝试的情况。 Toby Segaran[1] 提供了多个解释优化算法的案例，其中学生宿舍优化问题为根据人们的偏好分配有限资源的问题，如果延申该问题至城市空间规划领域，可以提出根据不同建设内容分配有限不同土地资源条件问题，例如根据约束条件设置，假设考虑到绿地、水体、生活服务、美食购物、教育培训、交通，建设成本等条件，存在5类不同条件的地块，

1. 地块A，绿地水体面积占比大，但其它条件偏弱，建设成本高；
2. 地块B，绿地水体面积占比偏中，有丰富的教育资源，交通条件良好，但生活服务、美食购物条件差，建设成本高；
3. 地块C，绿地水体面积占比偏中，生活服务、美食购物和交通条件好，但缺少教育资源，建设成本高；
4. 地块D，绿地少，无水体，无教育资源，但生活服务、美食购物和交通条件好，建设成本高中；
5. 地块E，裸地，建设成本低，除交通外，无其它资源。

对应变量为plots。

根据调研，地块A、B和C均有1个区域可选择，地块D和E有2个区域可选择，总共可选择区域数为7个，正好对应7个建设项目，每个建设项目给出首选和次选两个选择（对应变量为prefs），这很有可能发生用地配置冲突，那么如何协调分配，尽可能满足各建设项目的需求，面对存在多种甚至无数种解的问题，优化算法为从这些答案中寻找最优解提供了思路。

解决此问题的关键是，每个项目只能对应到一个地块区域和成本如何计算（定义目标函数）。Toby Segaran 在解决此问题时，给出了一个巧妙地解决思路，对于第1个问题，7个项目在选择地块时，采取逐一选择的策略，即第1个项目选择了一个地块后，随即将该地块从列表plots\_idx下移除，那么第2个项目选择时仅从剩余的地块中选择。此时，优化的“权重”列表（chromosome），实际上为用于选择剩余地块列表的索引值，这也解释了参数domain“权重”值区间的设定为什么为[(0, 6), (0, 5), (0, 4), (0, 3), (0, 2), (0, 1), (0, 0)]。对于第1个项目，此时有7个地块可以选择，因此区间均为(0,6)，而到第2个项目时，因为已经由第1个项目选择排除了一个，因此区间为(0,5)，以此类推。对于第2个问题，如果项目选择的地块为首选地块，则成本为0；如果为次选地块，则成本为1；如果均不在首选和次选中，则成本为3，因此成本值越小越趋于最优解。

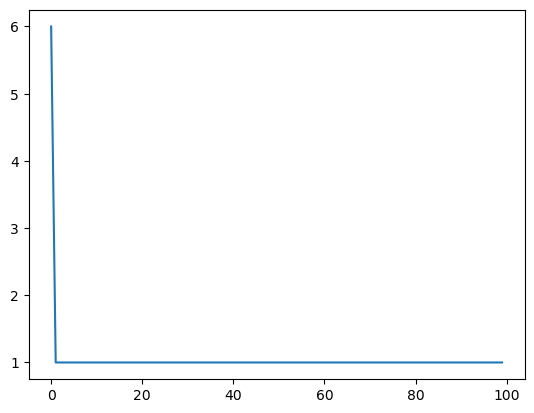
plots=['p\_a','p\_b','p\_c','p\_d','p\_e']  
prefs=[('s\_1',('p\_d','p\_c')),  
 ('s\_2',('p\_a','p\_e')),  
 ('s\_3',('p\_a','p\_e')),  
 ('s\_4',('p\_b','p\_a')),  
 ('s\_5',('p\_d','p\_b')),  
 ('s\_6',('p\_c','p\_e')),  
 ('s\_7',('p\_e','p\_b'))]  
  
domain=[(0,7-i-1) for i in range(0,7)]  
print(domain)  
  
# 构建目标函数  
def target\_function\_plot\_allocation(vec):  
 cost=0  
 plots\_idx=[0,1,2,3,3,4,4]  
 for i in range(len(vec)):  
 x=int(vec[i])  
 plot=plots[plots\_idx[x]]  
 pref=prefs[i][1]  
 if pref[0]==plot: cost+=0  
 elif pref[1]==plot: cost+=1  
 else: cost+=3  
   
 del plots\_idx[x]  
   
 return cost  
  
best\_score,epoch=usda\_heuristicsw.genetic\_algorithm\_SegarantT(domain,target\_function\_plot\_allocation,popsize=20,maxiter=100,verbose=20)

[(0, 6), (0, 5), (0, 4), (0, 3), (0, 2), (0, 1), (0, 0)]  
iter\_0: cost=6  
iter\_20: cost=1  
iter\_40: cost=1  
iter\_60: cost=1  
iter\_80: cost=1

打印对应迭代成本变化曲线。对于该实验，因为涉及的数据量小，快速收敛。

因为优化算法是趋于最优解，种群（population）的初始化，变异（mutation）和交叉（crossover）或配对（breeding）都应用了随机数生成随机基因（gene）的染色体（chromosome），筛选部分种群变异，从精英选拔（elitism）的染色体中随机选择父类通过交叉获取子类等过程，因此不同执行优化过程结果可能不同。

fig, ax=plt.subplots()  
ax.plot(epoch.keys(),epoch.values())  
plt.show()



因为“权重”列表为选择剩余地块的索引值，因此需要将计算结果返回为各个项目实际选择的地块。从上述计算结果看，7个项目中有6个保持了首选，有1个为次选，为一个不错的优化计算结果，能够满足绝大多项目的最佳需求。

def print\_solution(vec):  
 slots=[0,1,2,3,3,4,4]  
 # Loop over each students assignment  
 for i in range(len(vec)):  
 x=int(vec[i])  
 # Choose the slot from the remaining ones  
 plot=plots[slots[x]]  
   
 if plot in prefs[i][1]:  
 priority=prefs[i][1].index(plot)  
 else:  
 priority=-1  
 # Show the student and assigned dorm  
 print (f'{prefs[i][0]}: {plot}; priority: {priority}')  
 # Remove this slot  
 del slots[x]  
   
print(best\_score)   
print('-'\*50)  
print\_solution(best\_score)

[3, 4, 0, 0, 1, 0, 0]  
--------------------------------------------------  
s\_1: p\_d; priority: 0  
s\_2: p\_e; priority: 1  
s\_3: p\_a; priority: 0  
s\_4: p\_b; priority: 0  
s\_5: p\_d; priority: 0  
s\_6: p\_c; priority: 0  
s\_7: p\_e; priority: 0

根据解决问题的不同，在保持遗传算法基本思想不变的条件下，涉及到的选择、变异和交叉等具体内容时，会根据具体问题相应变化。Toby Segaran 提供的上述代码变异部分为对随机一个染色体的基因执行变化，如果该基因值大于给定区间（domain）最小值，且生成的[0,1]随机数小于0.5，则减去step步幅值；如果该基因值小于给定区间最大值则加上step步幅值。步幅值step根据“权重”配置确定，例如本例“权重”为地块选择索引值，因此配置值为1。交叉部分的方法根据随机一个整数值（区间为0到“权重”列表长度，即染色体的长度）交换染色体前后部分。在*权重决策*部分给出的遗传算法，为 Valdecy Pereira 就 MCDM 问题给出的适合解法，在选择、变异和交叉等计算上保持遗传算法基本思想不变条件下，与 Toby Segaran 提供的方法不同，具体可以查看*权重决策*部分的解释。

虽然 Toby Segaran 和 Valdecy Pereira 提供的遗传算法具体计算方式有所差异，核心仍旧是根据目标函数（计算成本值）寻找最优解的过程，因此将选择分配问题也以 Valdecy Pereira 提供的方法进行计算，参数的配置保持相同，虽然返回值为浮点数（可以调整原代码，转换为整数形式，适合此次问题形式），但取其整数后，其结果与 Toby Segaran 结果同（需要注意不同次运行优化算法，可能结果不同）。

domain\_min=[i[0] for i in domain]  
domain\_max=[i[1] for i in domain]  
best\_score\_2,epoch\_2=usda\_heuristicsw.genetic\_algorithm(population\_size=20,  
 mutation\_rate=0.2,  
 elite=4,  
 min\_values=domain\_min,  
 max\_values=domain\_max,  
 generations=100,  
 target\_function=target\_function\_plot\_allocation,  
 verbose=20)

Generation = 0 f(x) = 1.0  
Generation = 20 f(x) = 1.0  
Generation = 40 f(x) = 1.0  
Generation = 60 f(x) = 1.0  
Generation = 80 f(x) = 1.0  
Generation = 100 f(x) = 1.0

print(best\_score\_2[:-1])   
print('-'\*50)  
print\_solution(best\_score\_2[:-1])

[3.43977778 4.38075696 0.14946282 0.43850957 1.08662738 0.11878839  
 0. ]  
--------------------------------------------------  
s\_1: p\_d; priority: 0  
s\_2: p\_e; priority: 1  
s\_3: p\_a; priority: 0  
s\_4: p\_b; priority: 0  
s\_5: p\_d; priority: 0  
s\_6: p\_c; priority: 0  
s\_7: p\_e; priority: 0

### 3.3.1.2 复杂网络更新

Toby Segaran 提供的“绘制网络”的实验，虽然其目的是为了可视化网络结构时，避免绘制边线的交叉，但是由此可以拓展到复杂网络应用元启发式算法，在约束条件控制下，生成或更新网络的方法。这里给出一个探索性实验，将廊道和斑块（节点 ）的结构关系表述为复杂网络的形式，通过给定约束条件，1， 连通性；2，传递性；3，最近邻指数（点分布）；4. 避免边交叉，来生成一个根据约束条件可调的复杂网络形态。该实验分两个阶段，第1阶段为生成图；第2阶段为布局图。

#### 1) 生成图（廊道的连通性和传递性）

如果廊道具有较高的连通性（各个斑块之间有路线联系），斑块或节点之间形成环路或存在可替代路线，那么物种可以在廊道中有效移动，且避免廊道间隙、干扰、捕食者或者狩猎者的负面影响，提高移动效率[2]。对应连通性和环路，在*图属性的基本度量*一章提到NetworkX库的node\_connectivity方法可以计算顶点连通度（断开网络需要移除的最少边数）；transitivity方法可以计算传递性（为三角形（triangle）占三元（triads，为共享顶点的两个边）的比例）。生成图的思路借鉴了“学生宿舍优化问题”，基本流程为：

1. 通过指定顶点数nodes\_n，生成顶点（名称）列表ndoes，进而构建全部可成对的边pairs列表；
2. 指定生成图的边数edges\_n，根据边数构建索引值区间（“权重”取值范围）domain，方法同“学生宿舍优化问题”，只是从全部可成对的边pairs列表中选择部分边，例如下述实验边列表有36个边，而提取的边数为12；
3. 构建目标函数。根据提取的边构建图G，计算图的连通性和传递性，并通过分别被最大潜在连通性和值1减，将最大问题转换为最小问题；
4. 在连通性和传递性的贡献度分配上，增加了一个[0,1]区间的参数w来调整哪个指数具有更高的影响。

为了方便构建target\_function\_network()目标函数，一般先随机生成一个随机值（本例为索引值）列表edges\_idx，用于传入函数调试代码。

nodes\_n=9  
edges\_n=12  
w=0.7  
  
complete\_G=nx.complete\_graph(nodes\_n)  
connectivity\_complete=approx.node\_connectivity(complete\_G)  
print(connectivity\_complete)  
  
nodes=[f'n\_{i}' for i in range(nodes\_n)]  
pairs=list(itertools.combinations(nodes,2))  
print(len(pairs))   
  
domain=[(0,len(pairs)-i-1) for i in range(edges\_n)]  
print(domain)  
edges\_idx=[random.randint(i[0],i[1]) for i in domain] # 用于辅助构建目标函数而生成的随机“权重”值  
  
# 构建目标函数  
def target\_function\_network(edges\_idx):  
 global pairs,connectivity\_complete,w  
  
 pairs\_copy=copy.deepcopy(pairs)   
 edges=[]  
 for i in range(len(edges\_idx)):  
 x=int(edges\_idx[i])  
 pair=pairs\_copy[x]  
 edges.append(pair)  
 del pairs\_copy[x]  
  
 G=nx.Graph()  
 G.add\_edges\_from(edges)   
 connectivity\_=(connectivity\_complete-approx.node\_connectivity(G))/connectivity\_complete  
 transitivity\_=1-nx.transitivity(G)  
   
 cost=connectivity\_\*w+transitivity\_\*(1-w)  
  
 return cost  
  
cost=target\_function\_network(edges\_idx)   
print(cost)

8  
36  
[(0, 35), (0, 34), (0, 33), (0, 32), (0, 31), (0, 30), (0, 29), (0, 28), (0, 27), (0, 26), (0, 25), (0, 24)]  
0.7431818181818182

使用遗传算法寻找最优解。

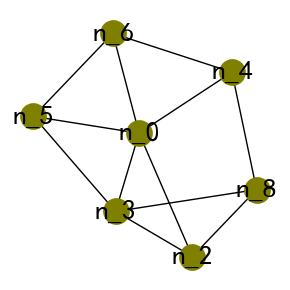
best\_idx\_1,\_=usda\_heuristicsw.genetic\_algorithm\_SegarantT(domain,target\_function\_network,step=1,popsize=50,maxiter=501,verbose=100)

iter\_0: cost=0.6449999999999999  
iter\_100: cost=0.5923387096774193  
iter\_200: cost=0.5923387096774193  
iter\_300: cost=0.5923387096774193  
iter\_400: cost=0.5923387096774193  
iter\_500: cost=0.5923387096774193

最优解为索引值列表，需要将其转换回对应的边，并构建图，打印图查看结果。

def show\_G\_solution(best\_idx):  
 global pairs  
  
 pairs\_copy=copy.deepcopy(pairs)   
 edges=[]  
 for i in range(len(best\_idx)):  
 x=int(best\_idx[i])  
 pair=pairs\_copy[x]  
 edges.append(pair)  
   
 del pairs\_copy[x]  
   
 G=nx.Graph()  
 G.add\_edges\_from(edges)  
 print(f'node connectivity={approx.node\_connectivity(G)};\ntransitivity={nx.transitivity(G)}')  
 usda\_network.G\_drawing(G)  
   
 return G  
   
G\_1=show\_G\_solution(best\_idx\_1)

node connectivity=3;  
transitivity=0.4838709677419355



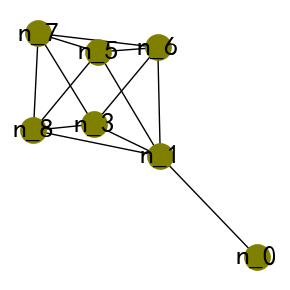
同样使用了 Valdecy Pereira 提供的遗传算法生成图，转换索引、打印查看图。

domain\_min=[i[0] for i in domain]  
domain\_max=[i[1] for i in domain]  
best\_idx\_2,\_=usda\_heuristicsw.genetic\_algorithm(  
 population\_size=50,  
 mutation\_rate=0.2,  
 elite=8,  
 min\_values=domain\_min,  
 max\_values=domain\_max,  
 generations=500,  
 target\_function=target\_function\_network,  
 verbose=100)

Generation = 0 f(x) = 0.6916666666666667  
Generation = 100 f(x) = 0.5006578947368421  
Generation = 200 f(x) = 0.5006578947368421  
Generation = 300 f(x) = 0.45  
Generation = 400 f(x) = 0.45  
Generation = 500 f(x) = 0.45

G\_2=show\_G\_solution(best\_idx\_2)

node connectivity=1;  
transitivity=0.6



#### 2) 布局图（最近邻指数和避免廊道交叉）

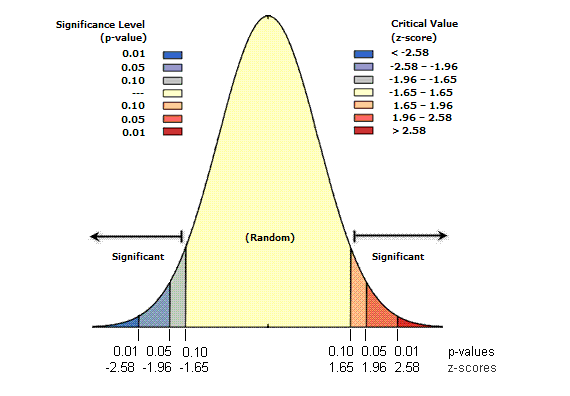
生成的图具有顶点和边，但是还没有为顶点指定地理空间位置。确定位置坐标的约束条件给了两个，一个是最近邻指数；再者为避免廊道交叉。

最近邻指数（Nearest Neighbor Index，NNI），计算各个点与其最近点的距离之和除以点数量，求得一个平均距离。如果该平均距离小于假设随机分布的平均距离值，则被分析的点分布呈聚类趋势；如果该平均距离大于假设分布的平均距离值，则被分析的点分布呈分散趋势。观测平均距离与预期平均距离的比值即为最近邻指数（或称为平均最近邻比率（average nearest neighbor ratio，ANN））。其计算公式为：， 式中，为观测平均距离，公式为：，为第个点与其最近点距离，为点数量；为预期平均距离，公式为：，为所有点最小外接矩形面积或者为指定的一个面积值。最近邻指数的统计标准差（z-score）为：，式中，。[3,4,5]

如果NNI（ANN）小于1，则点模式表现为集聚；如果该指数大于1，则趋于分散。

* 完全空间随机（complete spatial randomness，CSR），p-value与 z-score

在*空间自相关分析*部分解释邻接数时，确定两两一对单元BB（高值-高值），WW（低值-低值）分布是趋于集聚而不是完全随机分布时，通过生成完全空间随机（CSR）参照数据，构建一个参考分布来估计分析对象的统计意义。同样，对于NNI，采取了同样计算的思路，观察观测平均距离在由完全随机点数据集计算的预期平均距离分布上的位置，即由统计显著性水平（significance level）或临界值（critical value）分析 NNI 集聚或分散的显著性。p-value为一个概率，在地理空间模式分析（pattern analysis）中，为观测的空间模式由某个随机过程创建的概率。当p-value非常小时，意味着观测到的空间模式不太可能是随机的结果，因此拒绝原假设。z-score为一个标准差，可以通过标准差判断空间模式是否为小概率事件（例如空间点分布集聚、分散是否是随机结果等），p-value与 z-score的关系如图[6]：



避免廊道（边）交叉的基本思路是计算每条边交叉的边的分数值（fraction）。如果对于两条边，该分数介于0（边的一个端点）和1（边的另一端点）之间，则它们相互交叉；如果分数不在0和1之间，则边不交叉。下述迁移和调整了 Toby Segaran 的代码，并增加了NNI计算部分。因为避免边的交叉（total）和 NNI 值ann数量级存在差异，因此通过增加参数mu调整total值大小；为了控制NNI的集聚程度，增加了参数nu，公式为：CTAμ  
ν为对应调整的参数值。

# 构建目标函数  
def target\_function\_cross\_nni(coords):  
 global G,planning\_area  
   
 pts=list(G\_2.nodes)   
 links=list(G.edges)  
 # Convert the number list into a dictionary of pts:(x,y)  
 loc=dict([(pts[i],(coords[i\*2],coords[i\*2+1])) for i in range(0,len(pts))])  
 total=0  
   
 # Loop through every pair of links  
 for i in range(len(links)):  
 for j in range(i+1,len(links)):   
 # Get the locations   
 (x1,y1),(x2,y2)=loc[links[i][0]],loc[links[i][1]]  
 (x3,y3),(x4,y4)=loc[links[j][0]],loc[links[j][1]]  
  
 den=(y4-y3)\*(x2-x1)-(x4-x3)\*(y2-y1)   
 # den==0 if the lines are parallel  
 if den==0: continue   
   
 # Otherwise ua and ub are the fraction of the  
 # line where they cross  
 ua=((x4-x3)\*(y1-y3)-(y4-y3)\*(x1-x3))/den  
 ub=((x2-x1)\*(y1-y3)-(y2-y1)\*(x1-x3))/den   
  
 # If the fraction is between 0 and 1 for both lines  
 # then they cross each other  
 if ua>0 and ua<1 and ub>0 and ub<1:  
 total+=1   
   
 for i in range(len(pts)):  
 for j in range(i+1,len(pts)):   
 # Get the locations of the two nodes  
 (x1,y1),(x2,y2)=loc[pts[i]],loc[pts[j]]  
  
 # Find the distance between them  
 dist=math.sqrt(math.pow(x1-x2,2)+math.pow(y1-y2,2))  
 # Penalize any nodes closer than 50 pixels  
 if dist<50:  
 total+=(1.0-(dist/50.0))  
   
 ann,\_=usda\_network.nni(list(partition(2,coords)),planning\_area)   
 mu=1  
 nu=0.5  
 cost=pow(total,1/(mu+1))+abs(nu-ann)  
  
 return cost  
  
G=G\_2.copy()   
planning\_area=10000  
domain=[(0,100)]\*(len(list(G\_2.nodes) )\*2)  
print(domain)  
vec=[float(random.randint(domain[i][0],domain[i][1])) for i in range(len(domain))] # 用于辅助构建目标函数而生成的随机“权重”值  
  
target\_function\_cross\_nni(vec)

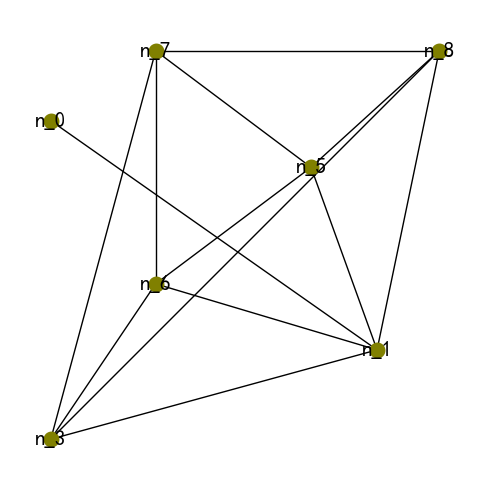
[(0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100), (0, 100)]  
  
  
  
  
  
7.726449995062749

best\_position,\_=usda\_heuristicsw.genetic\_algorithm\_SegarantT(domain,target\_function\_cross\_nni,popsize=50,maxiter=501,verbose=100)

iter\_0: cost=7.031376551186095  
iter\_100: cost=5.618358288334813  
iter\_200: cost=5.299805705965022  
iter\_300: cost=5.299805705965022  
iter\_400: cost=5.299805705965022  
iter\_500: cost=5.299805705965022

coords=list(partition(2,best\_position))  
pos=dict(zip(G.nodes, coords))  
print(pos)  
ann,\_=usda\_network.nni(coords,planning\_area)   
print(ann)  
  
usda\_network.G\_drawing(G,pos=pos,figsize=(5,5),node\_size=20,font\_size=15)

{'n\_6': (27, 40), 'n\_7': (27, 100), 'n\_1': (84, 23), 'n\_5': (67, 70), 'n\_8': (100, 100), 'n\_3': (0, 0), 'n\_0': (0, 82)}  
2.272290356842483



复杂网络更新实验是应用元启发式算法于复杂网络（图）的一次浅显探索，优化结果尚并不稳定

### 3.3.1.3 2维度布局优化——空间决策支持系统

地理空间模式一般为2维度的研究（目前存在少数3维度上的探索），由受2维度影响的约束条件构建目标函数及其返回值（成本值），但更新的对象（例如权重向量）不再是1维向量，而是2维矩阵，那么初始化的权重矩阵则由2维度升级为3维度。遗传算法是较容易由1维度跨到2维度的一种元启发式算法，染色体（chromosome）为2维度，由染色体构建的种群（population）为3纬度。就栅格数据而言，构成染色体的基因（gene）为每个栅格单元的某一属性值，例如土地利用、土地覆盖、空气质量、温湿度、经济统计指标、生态各类因子，或衍生及综合指数等，这就为地理空间模式的探索应用元启发式算法提供了可能。

对于空间规划（2纬度地理空间模式探索），为一种资源分配（resource allocation）问题，称为空间决策支持系统（spatial decision support system，SDSS），支持寻找土地利用的最佳空间分布，通常包含多准则决策（MCDM）技术，评估可用的替代解决方案，且此类问题通常为高度非线性（nonlinear），涉及大量数据和大相径庭的目标（objectives）。2000年左右就出现大量 SDSS 协助决策者解决此类问题[7,8,9]。其中Stewart, T. [10]、Janssen, R.[11]和Cao, K[12]在 SDSS 一脉相承，分别从方法提出（基于遗传算法），问题描述和公式构建，进一步系统化和实验验证，到遗传算法2维度上交叉与变异方式的改进，相对该领域类似的研究建立了更全面系统的方法体系，因此应用该体系阐释 SDSS，并基于 Python 再现实验。

#### 1）问题的数学描述和公式构建

为了理解公式描述，各个变量的定义字符和解释如下：

    栅格单元行索引（cell row index ），；

    栅格单元列索引（cell column index），；

    土地利用类型索引（land-use index），；

    （决策）准则目标索引（additive objective index），；（在 additive objective 翻译上，协同*权重决策*一章的多准则决策法用词）

    空间目标索引（spatial objective index），；

    第行，第列栅格单元的土地利用类型（specific land use for cell ()）；

    所有栅格单元土地利用类型分配的矢量表示（即，所有行和列的）（vector representation of land-use allocation for all cells（i.e.  for all and ））；

    二元决策变量，指示栅格单元是否被分配有第类土地利用类型（即，如果，则，否则）（ binary decision variable indicating whether cell is allocated to land use ）；

    分配有土地利用类型的栅格单元总数（total number of cells to be allocated to land use ）；

    分配有土地利用类型的最小栅格单元数（minimum number of cells to be allocated to land use ）；

    分配有土地利用类型的最大栅格单元数（maximum number of cells to be allocated to land use ）；

    分别为准则目标和空间目标函数的公式表述（formal representation additive and spatial objectives respectively）；

    准则目标的目标函数系数（objective function coefficient for additive objective ）;

    土地利用类型的聚类（集聚）数量（ numbers of clusters for land use ）；

    土地利用类型的最大聚类的相对大小（relative magnitude of the largest cluster for land use ）；

    土地利用类型的紧密度（compactness of land use ）；

    分别为准则目标和空间目标的理想值（ideal values for additive and spatial objectives respectively）；

    分别为准则目标和空间目标的目标值（靶值）（ goal values for additive and spatial objectives respectively）；

    将土地利用类型转换为类型的成本（ cost of changing land-use type into ）。

约束（constrains）条件包括3类，第1类为土地利用类型分配给各个栅格单元时的限制条件，例如一个栅格单元仅分配一种土地利用类型。或者特定的栅格单元仅允许某类土地利用类型存在等；第2类为准则目标（additive objective）的约束，例如每类土地利用类型的自然价值（收益/效益，benefits）、娱乐价值（收益）和一种土地类型转换为另一种类型的成本（成本，cost）等，成本或收益对应土地利用类型分配到各个栅格单元构成成本（收益）栅格（矩阵），总成本（收益）则为所有栅格单元之和，即累积相加（additively）;第3类为空间目标（spatial objective）的约束，例如各个土地利用类型在整个分析区域内的连通性、破碎化程度和紧密度等，通常与空间分布形态相关。

* 第1类约束——栅格数据的土地利用类型描述方法

分析区域为一个行，列的栅格，将第个土地利用类型是否分配到各栅格单元上时，存在两种情况，一种是分配，用值1表示；一种是不分配，用值0表示。即可以用一个二元决策变量表示，将第个土地利用类型分配给栅格单元时有；不分配则为，且每一栅格单元仅分配有一类土地利用类型，不可重叠，因此有，即对于同一栅格单元，对所有土地利用类型索引，只有一种土地利用类型值为，其余均为。

因为为二元决策变量，因此分配有土地利用类型的栅格单元总数为：。每一种土地利用类型分布总面积（即具有第类的土地利用类型栅格单元数）通常有限制，可表述为：，式中，为分配有土地利用类型的最小栅格单元数； 为分配有土地利用类型的最大栅格单元数。

* 第2类约束——决策目标

决策目标的约束可以表示为： ，式中为（决策）准则目标索引；为所有栅格单元土地利用类型分配的矢量表示。准则目标可以为成本，也可以为收益，为统一成本（最小化）和收益（最大化）的方向，统一按成本最小化计算，因此收益最大化需要转换方向，一种方式是对系数增加符号取反值；或者用收益的理想最大值减收益值。

* 第3类约束——空间目标

空间目标约束类型根据决策目的的差异会发生变化，一般而言包括三种：

1. 最小化每类土地利用类型的聚类数量，用表示。衡量了土地利用的碎片化程度，最小化，就是确保同类土地利用类型的栅格单元最大限度地连接在一起；
2. 最大化每类土地利用类型最大聚类地相对大小，用表示。如果，表明同一土地利用类型在地理空间分布上有多个聚类簇，最好使得其中一块簇的大小（连续栅格单元数量）占绝对优势，其它聚类簇尽量小；
3. 最大化每类土地利用类型的紧密度（compactness），用表示。通常定义为同一土地利用类型各聚类簇的周长（簇的沿边单元数）与簇大小（簇内所有单元数）的比值。

空间目标通常是高度非线性的，无法用封闭的数学形式（closed mathematical form）定义为的函数。 因此定义空间目标函数为，并统一约束方向为最小化，对于和需要取反值。

对于决策目标和空间目标的约束描述为最小化成本（靶值）的多目标优化问题，目标组合形式简单表示为：。简单的加法形式可能会导致具有极端倾向的高度偏差结果（一些目标得到很好的满足，而另一些则表现得很差），而不是平衡的妥协（balanced compromises ）。为了寻找更好的平衡妥协， 采用参考点（reference-point）（广义目标规划（generalized goal-programming））方法，涉及的关键点有：

1. 为每个目标指定理想值或，分别表示目标函数或实际上可以接受的最小可能值。某些情况，理想值可以直接计算，但有些情况需要启发式的评估；
2. 为每个目标指定靶值（目标值）或，表示可以归类为满意的表现水平，是对决策者的偏好进行建模的主要手段。该值通常并不是直接给出，而是通过给定理想值（ideal-value)和最差值（worst-value)，再给一个[0,1]的比率（nominal priority levels），获取理想值和最差值之间的一个值；
3. 通过最小化聚合函数（aggregate（scalarizing） function）：，尽可能接近指定的靶值。如果远离靶值，对应的惩罚项（指数）会使得成本值快速增大，因此避免了非常差的表现结果出现，从而产生平衡的妥协。

最小化聚合函数采取元启发式算法，该实验选择的为遗传算法。

#### 2）用Python 再现实验

**A - 目标约束配置和构建目标函数**

实验数据以 Janssen, R 的 Jisperveld 土地利用规划为例，覆盖面积为400 ha。该区域为西欧最大咸水泥炭草甸地区的一部分，整个区域水网纵横交错，存在多种稀有草甸鸟类和一些特殊植被。处理后的土地利用数据为 20×20 的栅格数据，包括9种土地利用类型，集约化农业（intensive agriculture）、粗放式农业（extensive agriculture）、住宅（residence）、工业（industry）、娱乐休闲（日游）（recreation（day trips））、娱乐休闲（过夜）（recreation（overnight））、湿地（wet natural area）、水体（娱乐用）（water（recreational use））和水体（限制进入）（water（limited access））等。该区域要开发一个结合娱乐休闲和自然价值的项目，根据自然保护的要求，提出了3条准则目标约束，

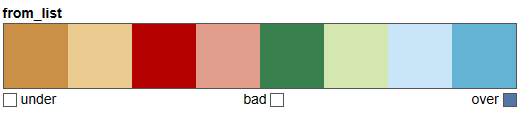
1. 最大化该区域的自然价值；
2. 最大化该区域的娱乐价值；
3. 最小化土地利用类型转换的成本。

同时，确定3个与规划相关的空间目标约束，

1. 最小化碎片化；
2. 最大化最大聚类簇；
3. 最大化紧密度。

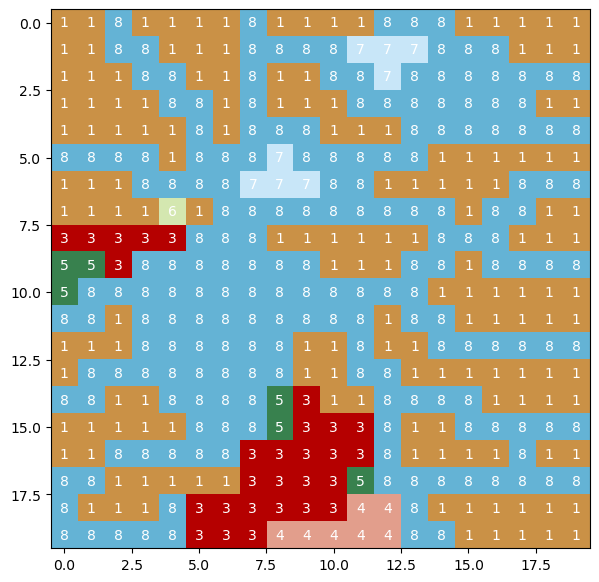
为了方便实验，该实验数据写入至USDA，通过load\_jisperveld\_data()方法调用。

import matplotlib  
  
lu\_class\_idNcolor={  
 'intensive\_agriculture':[1,"#ca9146"],  
 'extensive\_agriculture':[2,"#ebcb90"],  
 'residence':[3,"#b50000"],  
 'industrye':[4,"#e29e8c"],  
 'recreation\_day\_tripsy':[5,"#38814e"],  
 'recreation\_overnight':[6,"#d4e7b0"],  
 'wet\_natural\_area':[7,"#c8e6f8"],  
 'water\_recreational\_use':[8,"#64b3d5"],  
 'water\_limited\_access':[9,'#5475a8'],  
 }  
lu\_class\_color={v[0]:v[1] for v in lu\_class\_idNcolor.values()}  
cmap\_LC, norm=matplotlib.colors.from\_levels\_and\_colors(list(lu\_class\_color.keys()),list(lu\_class\_color.values()),extend='max')  
cmap\_LC



jisperveld\_data=usda\_datasets.load\_jisperveld\_data()  
jisperveld\_lu=jisperveld\_data['lu']  
jisperveld\_lu\_name=jisperveld\_data['lu\_name']  
print(jisperveld\_lu\_name)  
usda\_vis.imshow\_label2darray(jisperveld\_lu,figsize=(7,7),cmap=cmap\_LC,norm=norm,fontsize=10)

{1: 'intensive\_agriculture', 2: 'extensive\_agriculture', 3: 'residence', 4: 'industry', 5: 'recreation\_day\_trips', 6: 'recreation\_overnight', 7: 'wet\_natural\_area', 8: 'water\_recreational\_use', 9: 'water\_limited\_access'}



* 准则目标约束

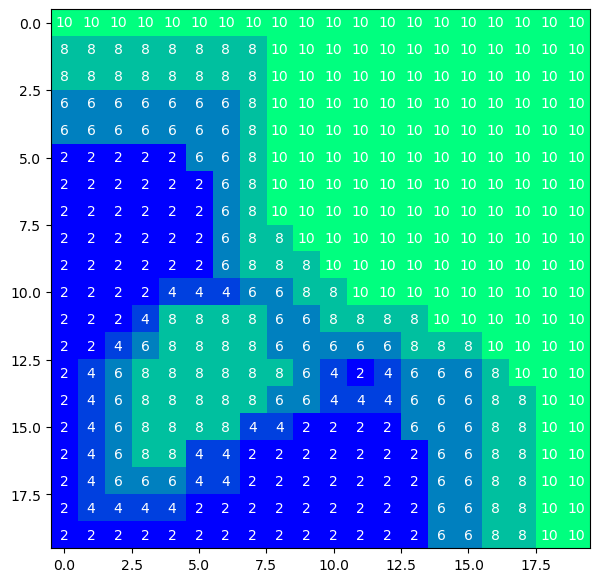
准则目标约束有3个，自然价值、娱乐休闲价值和土地利用转换，所采取的计算方法都是各个单元（土地利用类型）对应价值的总和。对于自然价值和娱乐休闲价值给出不同土地利用类型对应的值，一种方式是固定的值；另一种方式是成本地图的形式。固定的值如下，包括两列，nature\_value列用于自然价值计算；nature\_value列用于娱乐休闲价值计算。如果对应的值为字符串，例如nature\_vals、recreation\_b和recreation\_c，则为对应的成本地图，其值为所选土地利用类型所在位置对应的值。自然成本和娱乐休闲成本的公式可表述为：

jisperveld\_data['nature\_recreation\_vals']

|  | **nature\_value** | **recreational\_value** |
| --- | --- | --- |
| **intensive\_agriculture** | 4 | 6 |
| **extensive\_agriculture** | nature\_vals | nature\_vals |
| **residence** | 3 | 3 |
| **industry** | 1 | 1 |
| **recreation\_day\_trips** | 5 | recreation\_b |
| **recreation\_overnight** | 5 | recreation\_c |
| **wet\_natural\_area** | nature\_vals | 7 |
| **water\_recreational\_use** | 7 | recreation\_b |
| **water\_limited\_access** | nature\_vals | 1 |

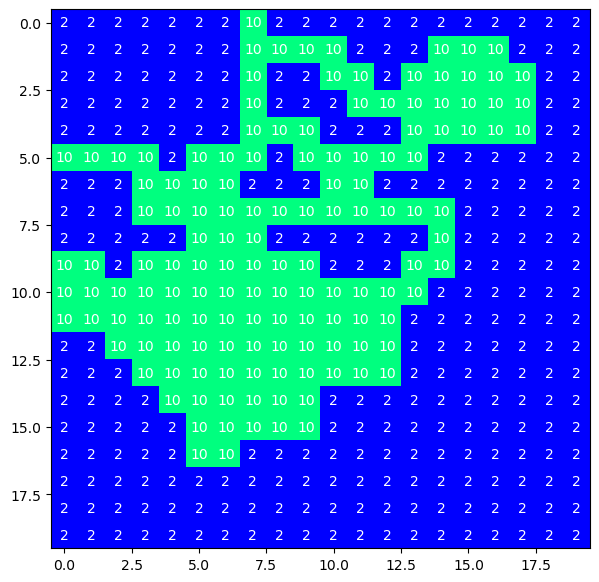
nature\_vals价值地图，为土地利用类型对草甸鸟类的价值，例如工业和住区对鸟类的干扰，其值较低，而粗放式农业、湿地和限制性水域具有较高的值。

usda\_vis.imshow\_label2darray(jisperveld\_data['nature\_vals'],figsize=(7,7),cmap='winter',fontsize=10)



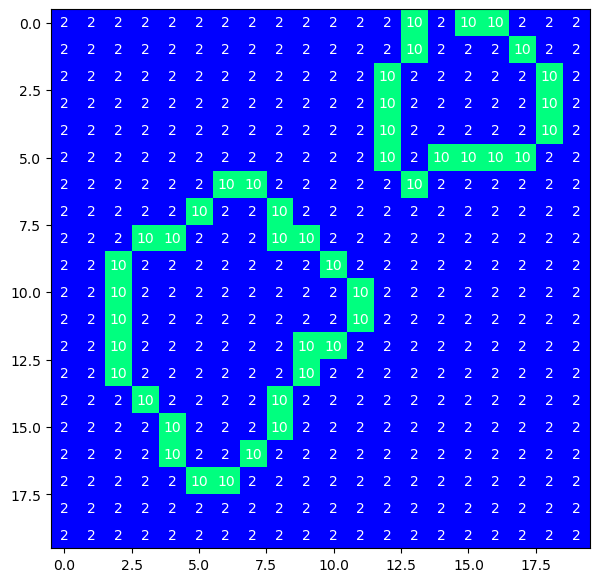
recreation\_b价值地图，反映了水域的价值。

usda\_vis.imshow\_label2darray(jisperveld\_data['recreation\_b'],figsize=(7,7),cmap='winter',fontsize=10)

  
png

recreation\_c价值地图，反映了水域边缘的价值。

usda\_vis.imshow\_label2darray(jisperveld\_data['recreation\_c'],figsize=(7,7),cmap='winter',fontsize=10)



土地利用转换成本是根据土地使用类型从一种转换为另一种导致的土地价值变化和转换相关措置（例如，管理成本，土壤转换为水的土方量等）的成本计算。如下表，空值为不能转换的类型，单位为欧元。土地利用类型转换成本公式表述为：。

jisperveld\_data['lu\_conversion\_cost']

|  | **intensive\_agriculture** | **extensive\_agriculture** | **residence** | **industry** | **recreation\_day\_trips** | **recreation\_overnight** | **wet\_natural\_area** | **water\_recreational\_use** | **water\_limited\_access** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **intensive\_agriculture** | 0.0 | -75.0 | 150.0 | 150.0 | -225.0 | 0.0 | -150.0 | -300.0 | -300.0 |
| **extensive\_agriculture** | 75.0 | 0.0 | 150.0 | 150.0 | -150.0 | 75.0 | -75.0 | -225.0 | -225.0 |
| **residence** | NaN | NaN | 0.0 | NaN | -10000.0 | -10000.0 | NaN | NaN | NaN |
| **industry** | NaN | NaN | NaN | 0.0 | -10000.0 | -10000.0 | NaN | NaN | NaN |
| **recreation\_day\_trips** | 150.0 | 75.0 | 3.0 | 300.0 | 0.0 | 150.0 | 0.0 | -150.0 | -150.0 |
| **recreation\_overnight** | 0.0 | -75.0 | 150.0 | 150.0 | -150.0 | 0.0 | -150.0 | -300.0 | -230.0 |
| **wet\_natural\_area** | NaN | 75.0 | 225.0 | 225.0 | -75.0 | 150.0 | 0.0 | -75.0 | -75.0 |
| **water\_recreational\_use** | 100.0 | 100.0 | NaN | NaN | NaN | NaN | 0.0 | 0.0 | 15.0 |
| **water\_limited\_access** | 100.0 | 100.0 | NaN | NaN | NaN | NaN | 0.0 | 0.0 | 0.0 |

定义target\_function\_additive\_objectives\_constrains()目标函数，实现准则目标成本（收益）值的计算。

def target\_function\_additive\_objectives\_constrains(lus):  
 global jisperveld\_data   
   
 nature\_recreation\_vals=jisperveld\_data['nature\_recreation\_vals']   
 lu\_name=jisperveld\_data['lu\_name']  
 nature\_vals\_lst=[]  
 recreation\_vals\_lst=[]   
   
 for k,v in lu\_name.items():   
 lu\_mask=lus==k  
   
 # natural cost  
 nature\_val=nature\_recreation\_vals['nature\_value'][v]   
 if type(nature\_val)==str:  
 nature\_vals\_map=jisperveld\_data[nature\_val]  
 lu\_nature\_val=nature\_vals\_map\*lu\_mask   
 else:  
 lu\_nature\_val=lu\_mask\*nature\_val  
   
 nature\_vals\_lst.append(lu\_nature\_val)  
   
 # recreational cost  
 recreation\_val=nature\_recreation\_vals['recreational\_value'][v]   
 if type(recreation\_val)==str:  
 recreation\_vals\_map=jisperveld\_data[recreation\_val]  
 lu\_recreation\_val=recreation\_vals\_map\*lu\_mask   
 else:  
 lu\_recreation\_val=lu\_mask\*recreation\_val  
   
 recreation\_vals\_lst.append(lu\_recreation\_val)   
   
 # lu conversion cost  
 lus\_original=jisperveld\_data['lu']  
 changed\_mask=~((lus-lus\_original)==0)  
 original=lus\_original[changed\_mask]  
 changed=lus[changed\_mask]  
 original2changed=list(zip(original,changed))  
   
 conversion\_cost\_matrix=jisperveld\_data['lu\_conversion\_cost'].T  
 conversion\_cost\_matrix.fillna(9999999999,inplace=True) # 对空值进行替换  
   
 conversion\_cost=0  
 for pair in original2changed:  
 conversion\_pair\_cost=jisperveld\_data['lu\_conversion\_cost'].T[lu\_name[pair[0]]][lu\_name[pair[1]]]  
 conversion\_cost+=conversion\_pair\_cost   
   
 nature\_costs=np.array(nature\_vals\_lst).sum(axis=0)  
 recreation\_costs=np.array(recreation\_vals\_lst).sum(axis=0)  
   
 nature\_cost=nature\_costs.sum()  
 recreation\_cost=recreation\_costs.sum()  
   
 return nature\_cost,recreation\_cost,conversion\_cost  
  
jisperveld\_lu\_changed=jisperveld\_lu+(jisperveld\_lu==3)\*3 # 测试用假设已变化的土地利用类型  
nature\_cost,recreation\_cost,conversion\_cost=target\_function\_additive\_objectives\_constrains(jisperveld\_lu\_changed)  
print(nature\_cost,recreation\_cost,conversion\_cost)

2239 2493 -280000.0

* 空间目标约束

空间目标约束有3个，为碎片化、最大簇比率和紧密度。碎片化的计算是使用components-3d库connected\_components方法计算连通域标签，获得具有同一土地利用类型所有簇的数量。最大簇比率是由连通域标签得出同一土地利用类型的所有簇大小，由最大值除以所有属于该土地利用类型的栅格单元数获得。紧密度计算方法做了调整，按一个基因与邻里8个基因对形成不同组合的数量除以8的方式，求和所有不同组合基因对并除以栅格总数。

定义target\_function\_spatial\_objectives\_constrains()函数，实现空间目标成本（收益）值计算。

def target\_function\_spatial\_objectives\_constrains(lus):   
 # minimize fragmentation  
 clump\_2darray,C=cc3d.connected\_components(lus,connectivity=8,return\_N=True,out\_dtype=np.uint64)   
   
 # maximize the largest cluster  
 unique, counts=np.unique(clump\_2darray, return\_counts=True)  
 unique\_counts=dict(zip(unique, counts))  
 class\_clump=np.stack((lus,clump\_2darray),axis=2)  
 class\_clump\_mapping=usda\_signature.lexsort\_based(class\_clump.reshape(-1,2)).tolist()  
 class\_clump\_mapping.sort(key=lambda x:x[0])  
 class\_clump\_max=[[int(i[0]),unique\_counts[i[1]]] for i in class\_clump\_mapping]  
 cluster\_num\_dict=defaultdict(list)   
 for k,v in class\_clump\_max:  
 cluster\_num\_dict[k].append(v)  
 L\_k={k:max(v)/sum(v) for k,v in cluster\_num\_dict.items()}  
 L=sum(L\_k.values())  
  
 # maximize compactness  
 xextent,yextent=lus.shape  
 ngh\_finder=usda\_signature.GridNghFinder(0, 0, xextent-1,yextent-1)  
 x\_=np.linspace(0, xextent-1, xextent)  
 y\_=np.linspace(0, yextent-1, yextent)  
 x\_idx, y\_idx=np.meshgrid(x\_, y\_)   
 xy=np.stack((x\_idx,y\_idx),axis=2).reshape(-1,2).astype(int)  
 pairs=np.empty((0,2),int)  
 R=0  
 for i in xy:  
 nghs=ngh\_finder.find(i[0],i[1])  
 ngh\_vals=[lus[j[0],j[1]] for j in nghs]  
 i\_val=lus[i[0],i[1]]  
 ngh\_vals.remove(i\_val)  
 i\_pairs=np.array([[i\_val,k] for k in ngh\_vals])  
 i\_pairs\_set=set(list(zip(i\_pairs.T[0],i\_pairs.T[1])))   
 pairs\_num\_fraction=len(i\_pairs\_set)/8  
 R+=pairs\_num\_fraction/400  
   
 return C,L,R  
   
C,L,R=target\_function\_spatial\_objectives\_constrains(jisperveld\_lu)   
print(C,L,R)

29 4.893994540491356 0.26937499999999776

* 土地利用类型的面积（栅格单元数）控制约束

规格过程中通常要限制各类土地利用类型的面积，其成本值的计算方式为： Nk 为分配有土地利用类型的栅格单元总数；为分配有土地利用类型的最小栅格单元数；为分配有土地利用类型的最大栅格单元数； 为比例因子，本实验配置为栅格总数。如果土地利用类型栅格单元数位于给定区间，则返回成本值为0；如果小于或者大于，则返回到或者的距离值，并除以栅格总数。这使得土地利用类型栅格单元数或多或少的在指定区间内，而不是趋向于一个具体的值，为趋向于不精确的确定。

定义target\_function\_lu\_area()函数实现土地利用类型的面积约束成本计算。

def target\_function\_lu\_area(lus):  
 global domain\_lu\_area,jisperveld\_data   
   
 lu\_name=jisperveld\_data['lu\_name']  
 lu\_idx2name={v:k for k,v in lu\_name.items()}  
 domain\_lu\_area\_idx={lu\_idx2name[k]:v for k,v in domain\_lu\_area.items()}  
 unique, counts=np.unique(lus, return\_counts=True)  
 unique\_counts=dict(zip(unique, counts))  
   
 minimumBound\_onClusterSize=1  
 p=4  
   
 cost=0  
 for idx,count in unique\_counts.items():   
 domain=domain\_lu\_area\_idx[idx]  
 dis=max([max(0,domain[0]-count),max(0,count-domain[1])])  
 dis\_abs\_fraction=abs(dis)/minimumBound\_onClusterSize # lus.size  
 cost+=pow(dis\_abs\_fraction,p)   
   
 return cost  
  
domain\_lu\_area={'intensive\_agriculture':[80,150],  
 'extensive\_agriculture':[20,65],  
 'residence':[20,45],   
 'industry':[5,15],   
 'recreation\_day\_trips':[0,70],   
 'recreation\_overnight':[0,35],  
 'wet\_natural\_area':[0,30],  
 'water\_recreational\_use':[120,150],  
 'water\_limited\_access':[0,60]}  
   
lu\_area\_cost=target\_function\_lu\_area(jisperveld\_lu)  
print(lu\_area\_cost)

3421202.0

* 构建聚合目标函数

上述计算了三大类约束，准则目标约束（含3小类）、空间目标约束（含3小类）和土地利用类型的面积约束。各小类约束的成本值区间范围不同，约束方向不同，通过采用前文阐释的参考点（reference-point）方法，可以标准化数值和方向，定义domain\_objectives\_worst2best变量存储各类约束的最小值和最大值（理想值），靶值的比率由参数priority\_level调整，实验中配置值为0.5，即取最小值和最大值的中间值作为靶值。土地利用类型的面积约束不使用参考点的方法，直接使用*土地利用类型的面积（栅格单元数）控制约束*部分的计算公式。

对各类约束成本区间范围的指定，可以根据约束条件推断得出，例如自然成本区间natrue\_cost和娱乐休闲成本区间recreation\_cost根据前文给出的自然价值、娱乐休闲价值表推断，最小价值为1，最大价值为10，栅格总数为400，因此区间为[400,4000]；对于土地利用转换的成本区间lu\_conversion\_cost根据前文土地利用类型转换成本表推断，用转换成本最小值（-10000）和最大值（300）乘以栅格总数；对于最小化每类土地利用类型的聚类数量成本区间C，因为总共有9类土地利用类型，如果每类只有一个簇，则为9，假设各为5个，则有45个，因此估计区间为[45,9]；对于最大化每类土地利用类型最大聚类地相对大小区间L，一类土地类型只有1个簇的情况值为1，因此最大值为9，假设最大簇比率比率为0.2，则区间为[1.8,9]；对于最大化每类土地利用类型的紧密度区间R，自身值域为[0,1]。

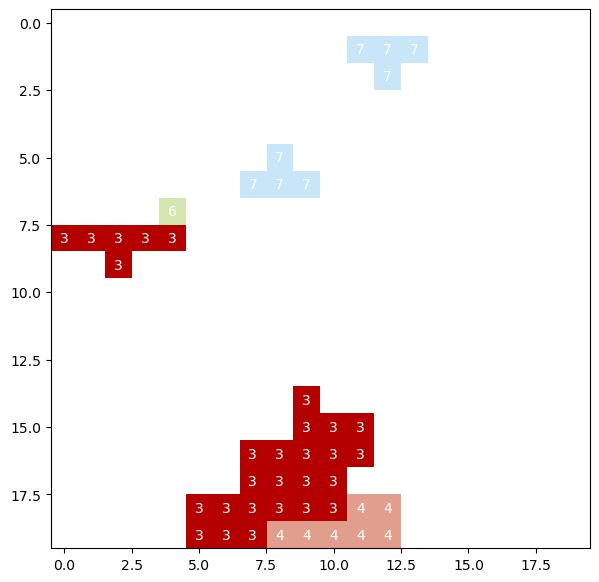
def target\_function\_jisperveld\_plan(lus):  
 global domain\_objectives\_worst2best,cost\_filter  
  
 nature\_cost,recreation\_cost,conversion\_cost=target\_function\_additive\_objectives\_constrains(lus)  
 C,L,R=target\_function\_spatial\_objectives\_constrains(lus)   
 cost\_dict\_={'natrue\_cost':nature\_cost,'recreation\_cost':recreation\_cost,'lu\_conversion\_cost':conversion\_cost,'C':C,'L':L,'R':R,'lu\_area\_cost':None}   
 cost\_dict={k:v for k,v in cost\_dict\_.items() if k in cost\_filter}   
   
 p=4   
 cost=0   
   
 def reference\_pt(k,priority\_level):   
 domain=domain\_objectives\_worst2best[k]   
 target\_val=(domain[1]-domain[0])\*priority\_level+domain[0]  
 g\_v=(v-domain[1])/(target\_val-domain[1])  
 g\_v\_power=pow(g\_v,p)  
   
 return g\_v\_power  
   
 for k,v in cost\_dict.items():  
 if k=='lu\_area\_cost':  
 lu\_area\_cost=target\_function\_lu\_area(lus)   
 cost+=lu\_area\_cost   
 elif k in ['natrue\_cost','recreation\_cost','lu\_conversion\_cost']:  
 g\_v\_power=reference\_pt(k,0.5)  
 cost+=g\_v\_power  
 elif k in ['C','L','R']:  
 g\_v\_power=reference\_pt(k,0.5)  
 cost+=g\_v\_power   
  
 return cost   
   
domain\_objectives\_worst2best={'natrue\_cost':[1\*400,10\*400],'recreation\_cost':[1\*400,10\*400],'lu\_conversion\_cost':[300\*400,-10000\*400],'C':[5\*9,1\*9],'L':[0.2\*9,1\*9],'R':[1,0]}   
cost\_filter=['natrue\_cost','recreation\_cost','lu\_conversion\_cost','C','L','R','lu\_area\_cost']  
cost=target\_function\_jisperveld\_plan(jisperveld\_lu\_changed)  
print(cost)

17.62373549948494

* 固定栅格单元的土地利用类型

更新规划中会存在不可变动的土地利用类型区域（栅格单元），也可以为预先定义的栅格单元土地利用类型，来排除不可能或不能接受的土地利用类型转换。下图显示了固定栅格的土地利用类型分布。固定栅格单元的土地利用类型不作为约束传入目标函数，需要在遗传算法过程中加入该条件，在初始化种群，交叉和变异染色体后都需要用该条件置换对应栅格的土地利用类型。

usda\_vis.imshow\_label2darray(jisperveld\_data['fixed\_LU'],figsize=(7,7),cmap=cmap\_LC,norm=norm,fontsize=10)

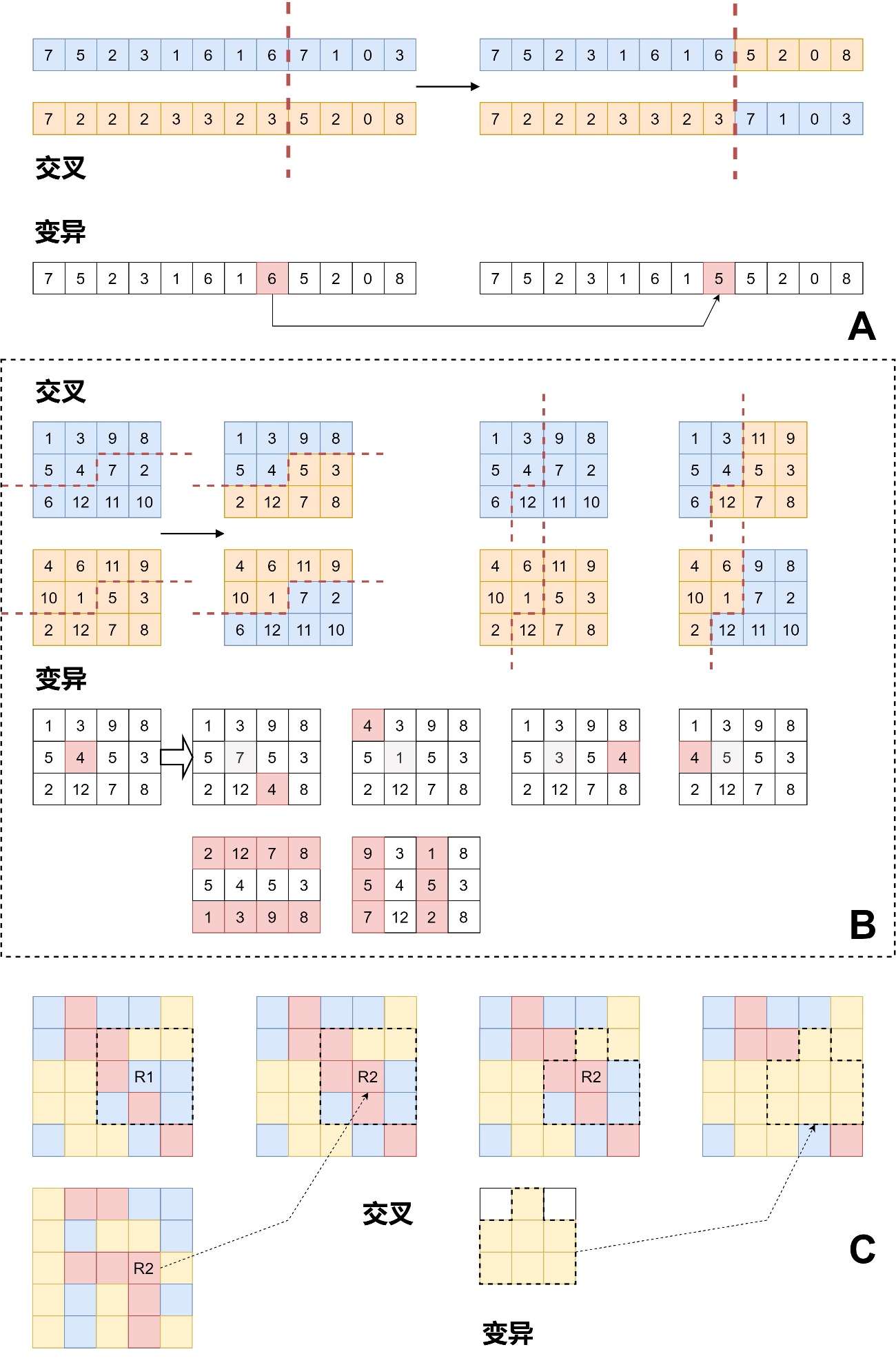


**B - 2维度染色体的交叉和变异**

遗传算法基本的交叉和变异（1维度）如下图A[1]所示。交叉是随机获取一个交叉点，将父代（2个染色体）按照交叉点互换生成子代。变异则是随机从染色体中提取一个基因，通常按基因变化的幅度（“权重”值变化的幅度）增减值。

当染色体从1维到2维，交叉和变异的算法样式较为丰富，通常需要根据优化对象采取适宜的方式。下图B[13]为一个2维染色体较为简单的变化，其交叉是根据随机交叉点水平或垂直置换父代2个染色体对应位置的基因。变异则是随机提取一个基因，将其与其余的一个随机基因置换；或者随机行或列的置换。此类2维度染色体的变异和交叉并未体现空间属性，例如土地利用或土地覆盖类型通常具有集聚的特点等，因此对具有空间属性的数据寻找最优解时，执行此类交叉和变异的方法通常很难达到预期结果。但是此类算法可解决不具有空间属性的问题，例如将飞机分配到时间表的调度问题，矩阵的横轴为时间，纵轴为可用飞机序号，不同架飞机的飞行任务需要被排入时间表中，约束条件有周转时间约束，连续飞行的抵达和飞离的位置应形同，及餐饮和燃料成本等。

为了能够优化多目标约束的土地利用分配等问题，使2维度染色体的交叉和变异时考虑到增加空间的连接度、紧密度、单类最大簇相对大小和减小碎片化等问题，Cao, K等提出了基于边界的交叉算子（ Boundary-based Crossover Operator，CBO）和基于补丁的变异算子（Patch-based Mutation Operator，MPO），及基于边界的变异算子（Boundary-based Muta- tion Operator，MBO），如下图C[12]。CBO交叉的方法是，对于从父代2个染色体随机提取的基因R1和R2（位置同），如果两个基因不同（即土地利用类型不同），且R1的邻里8个单元中有与R2相同的基因，则将R1置换为R2。如果随机提取的同一位置的R1和R2基因相同，则继续寻找，直至找到不同的位置进行交叉。该方法增加了相同基因的集聚可能。MPO变异的方法是，构建一个变异窗口，即从9个方形单元格中随机选择7个连续的单元格，并随机选择一种基因。然后从染色体中随机选择变异位置，例如R2基因，将其对应变异窗口位置的基因进行基因替换。MBO变异基本同MPO变异，只是从染色体中随机提取的基因R2的8个邻里单元中需要有同变异窗口相同的基因才执行变异操作。上述变异的方式同样增加了同一基因集聚的可能，并将原染色体可能破碎的区域替换为同类基因连续的团块。



将不同方法的交叉和变异增加到遗传算法genetic\_algorithm\_2d\_fixed\_map程序中，通过参数crossover\_name调整交叉算法（例如，crossover\_CBO、crossover\_tsai等）；通过参数mutation\_nam调整变异算法（例如，mutation\_MPO\_MBO（配合参数mpo\_mbo）、mutation\_tsai\_2和mutation\_tsai\_1等）。

程序中算法的命名部分以作者定义的算法名称组成，部分为作者的姓名组成

**C - 寻找多目标空间规划的最优解**

完成约束的定义，目标函数的定义，并增加了部分交叉和变异算法，试图寻找多目标空间规划的最优解。元启发式算法通常需要不断地调试参数值，甚至调整选择、交叉和变异的算法，才能够较好的拟合，获取可以参考的最优解，这对于解算非线性的空间问题尤其突出。初始化种群时，一般为完全随机生成，通过寻找成本值趋近于0（最小化）适合的染色体，找到最优解。对于已有现状条件的土地利用，也尝试初始化的种群为现状条件，在已有基础上进行扰动寻找最优解，配置的参数为population\_init。

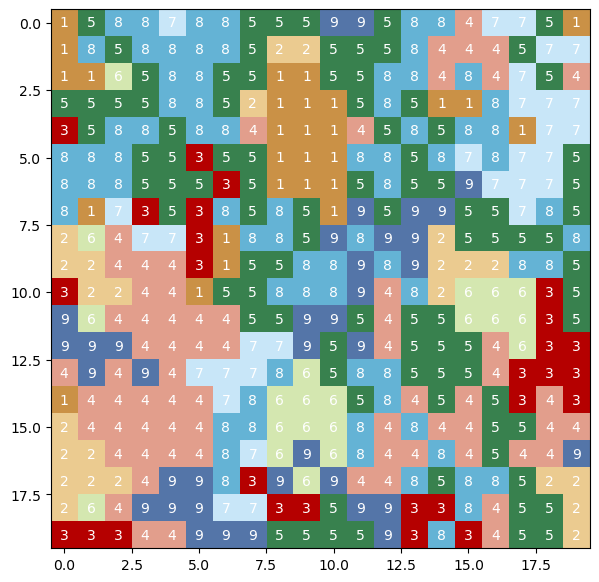
为了检验约束设计及其遗传算法收敛情况，首先进行了单目标的测试。

1. 最小化碎片化检验

最小化碎片化是计算连通域标签，获得总共簇的数量，并尽量使得该值最小。通过下述计算，对于一个随机生成的种群（50个染色体），经过1000次的迭代，迭代的最优解得以下降，最后最优解的连通域簇数量为63，表明通过连通域计算簇数的方法对于降低碎片化可行，如果继续迭代，该值可能继续下降。

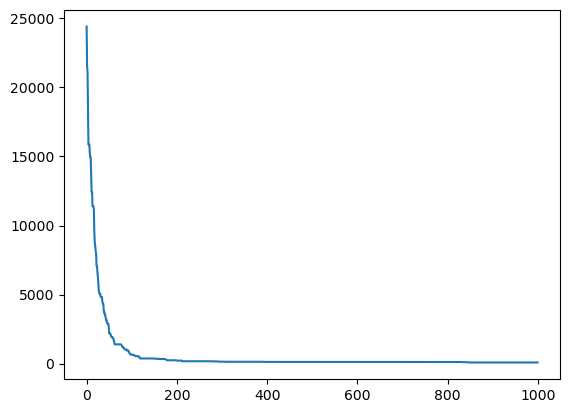
cost\_filter=['C']  
objects\_idx=list(range(1,10))  
rows\_n=20  
cols\_n=20   
fixed\_map=jisperveld\_data['fixed\_LU']  
  
domain\_lu\_area={'intensive\_agriculture':[80,150],  
 'extensive\_agriculture':[20,65],  
 'residence':[20,45],   
 'industry':[5,15],   
 'recreation\_day\_trips':[0,70],   
 'recreation\_overnight':[0,35],  
 'wet\_natural\_area':[0,30],  
 'water\_recreational\_use':[120,150],  
 'water\_limited\_access':[0,60]}  
  
domain\_objectives\_worst2best={'natrue\_cost':[1\*400,10\*400],'recreation\_cost':[1\*400,10\*400],'lu\_conversion\_cost':[300\*400,-10000\*400],'C':[5\*9,1\*9],'L':[0.2\*9,1\*9],'R':[1,0]}   
  
pattern\_generated\_random\_C,epoch=usda\_heuristicsw.genetic\_algorithm\_2d\_fixed\_map(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 population\_size=50,  
 generations=1000,  
 mutation\_rate=0.5,  
 target\_function=target\_function\_jisperveld\_plan,  
 crossover\_name='crossover\_CBO',  
 mutation\_name='mutation\_MPO\_MBO',   
 mpo\_mbo='mpo',   
 verbose=100)  
usda\_vis.imshow\_label2darray(pattern\_generated\_random\_C,figsize=(7,7),cmap=cmap\_LC,norm=norm,fontsize=10)

Generation = 0; f(x) = 24414.0625  
Generation = 100; f(x) = 653.2441796220087  
Generation = 200; f(x) = 215.92669753086423  
Generation = 300; f(x) = 140.75918305136415  
Generation = 400; f(x) = 115.42982205456484  
Generation = 500; f(x) = 115.42982205456484  
Generation = 600; f(x) = 115.42982205456484  
Generation = 700; f(x) = 115.42982205456484  
Generation = 800; f(x) = 115.42982205456484  
Generation = 900; f(x) = 81.0



C,\_,\_=target\_function\_spatial\_objectives\_constrains(pattern\_generated\_random\_C)   
print(C)  
  
fig, ax=plt.subplots()  
ax.plot(epoch.keys(),epoch.values())  
plt.show()

63

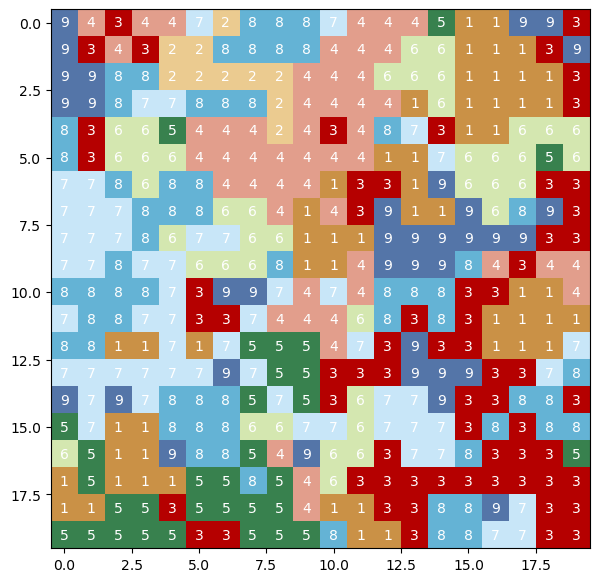


1. 最大化最大聚类簇

最大化最大聚类簇是各土地利用类型最大簇与该类所有栅格单元的比值。随机初始化的种群，迭代1000次，成本值稳步下降，表明该空间目标可以增大单个土地利用类型的最大簇相对大小。

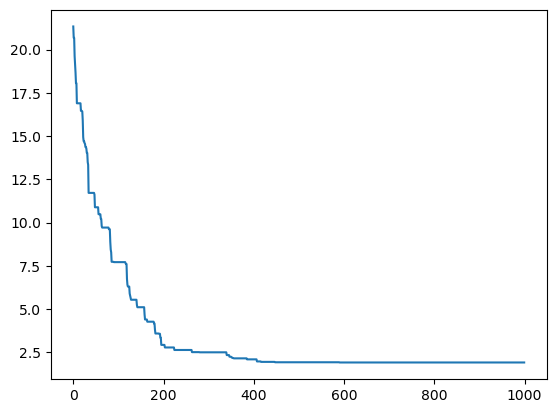
cost\_filter=['L']  
  
pattern\_generated\_random\_L,epoch=usda\_heuristicsw.genetic\_algorithm\_2d\_fixed\_map(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 population\_size=50,  
 generations=1000,  
 mutation\_rate=0.5,  
 target\_function=target\_function\_jisperveld\_plan,  
 crossover\_name='crossover\_CBO',  
 mutation\_name='mutation\_MPO\_MBO',   
 mpo\_mbo='mpo',  
 verbose=100)  
usda\_vis.imshow\_label2darray(pattern\_generated\_random\_L,figsize=(7,7),cmap=cmap\_LC,norm=norm,fontsize=10)

Generation = 0; f(x) = 21.3395847255005  
Generation = 100; f(x) = 7.715728408488455  
Generation = 200; f(x) = 2.930991796196175  
Generation = 300; f(x) = 2.502192278872799  
Generation = 400; f(x) = 2.095008764627904  
Generation = 500; f(x) = 1.926847709156549  
Generation = 600; f(x) = 1.9143194758101802  
Generation = 700; f(x) = 1.9143194758101802  
Generation = 800; f(x) = 1.9143194758101802  
Generation = 900; f(x) = 1.9143194758101802



\_,L,\_=target\_function\_spatial\_objectives\_constrains(pattern\_generated\_random\_L)   
print(L)  
  
fig, ax=plt.subplots()  
ax.plot(epoch.keys(),epoch.values())  
plt.show()

4.765461310579871

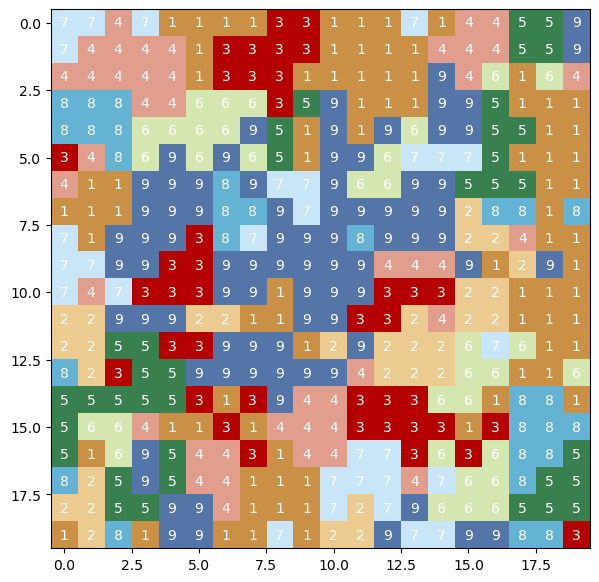


1. 最大化紧密度

通过检验，迭代1000次，成本值不断下降，表明最小化基因与邻里8个基因对形成不同组合的数量的方法可以增加基因的紧密度。

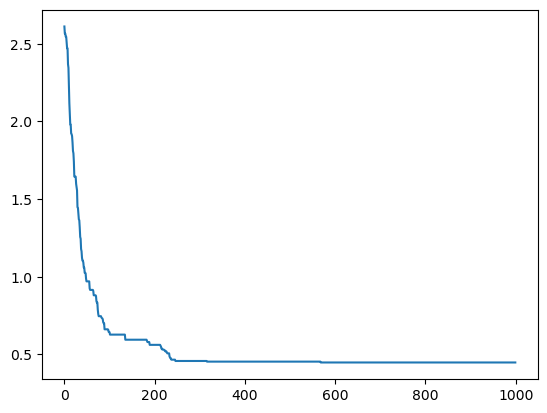
cost\_filter=['R']  
  
pattern\_generated\_random\_R,epoch=usda\_heuristicsw.genetic\_algorithm\_2d\_fixed\_map(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 population\_size=50,  
 generations=1000,  
 mutation\_rate=0.5,  
 target\_function=target\_function\_jisperveld\_plan,  
 crossover\_name='crossover\_CBO',  
 mutation\_name='mutation\_MPO\_MBO',   
 mpo\_mbo='mpo',  
 verbose=100)  
usda\_vis.imshow\_label2darray(pattern\_generated\_random\_R,figsize=(7,7),cmap=cmap\_LC,norm=norm,fontsize=10)

Generation = 0; f(x) = 2.6117034558618255  
Generation = 100; f(x) = 0.6434352223390067  
Generation = 200; f(x) = 0.561460441684709  
Generation = 300; f(x) = 0.45766070628905064  
Generation = 400; f(x) = 0.4535017567384239  
Generation = 500; f(x) = 0.4535017567384239  
Generation = 600; f(x) = 0.4480006618287534  
Generation = 700; f(x) = 0.4480006618287534  
Generation = 800; f(x) = 0.4480006618287534  
Generation = 900; f(x) = 0.4480006618287534



\_,L\_,R=target\_function\_spatial\_objectives\_constrains(pattern\_generated\_random\_R)   
print(R)  
  
fig, ax=plt.subplots()  
ax.plot(epoch.keys(),epoch.values())  
plt.show()

0.40906249999999716

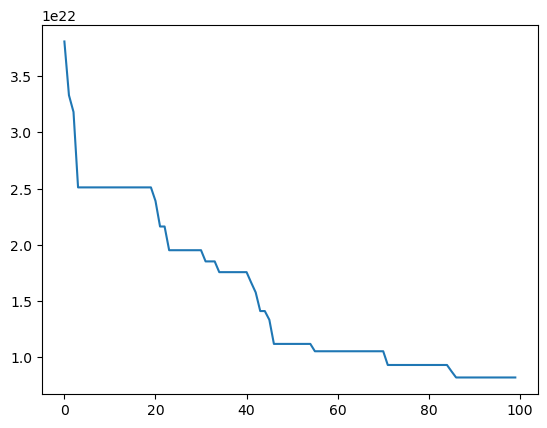


1. 准则目标约束检测

对3个准则目标同时进行检测。随迭代的增加，成本值能够持续下降，表明前文对准则目标的约束配置是适合的。

cost\_filter=['natrue\_cost','recreation\_cost','lu\_conversion\_cost']  
  
pattern\_generated\_random\_additive,epoch=usda\_heuristicsw.genetic\_algorithm\_2d\_fixed\_map(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 population\_size=50,  
 generations=100,  
 mutation\_rate=0.5,  
 target\_function=target\_function\_jisperveld\_plan,  
 crossover\_name='crossover\_CBO',  
 mutation\_name='mutation\_MPO\_MBO',   
 mpo\_mbo='mpo',  
 verbose=10)  
  
fig, ax=plt.subplots()  
ax.plot(epoch.keys(),epoch.values())  
plt.show()

Generation = 0; f(x) = 3.808062600912729e+22  
Generation = 10; f(x) = 2.5106993425899837e+22  
Generation = 20; f(x) = 2.3904481649399735e+22  
Generation = 30; f(x) = 1.952102317623157e+22  
Generation = 40; f(x) = 1.7570537172036949e+22  
Generation = 50; f(x) = 1.1190267994338828e+22  
Generation = 60; f(x) = 1.0537004297430581e+22  
Generation = 70; f(x) = 1.0537002757759955e+22  
Generation = 80; f(x) = 9.316688413940868e+21  
Generation = 90; f(x) = 8.205569806348615e+21

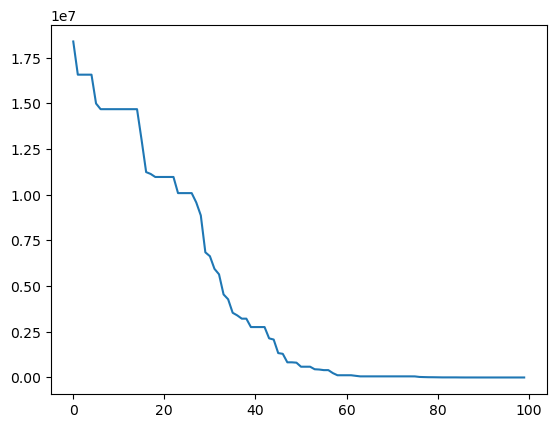


1. 土地利用面积约束检测

由下述计算结果，持续的成本值收敛，及各个土地利用类型数量统计（统计结果均位于初始配置区间内），可以初步判断，前文对土地利用面积区间约束的计算方法可以逐迭代的降低成本值，具有可行性。

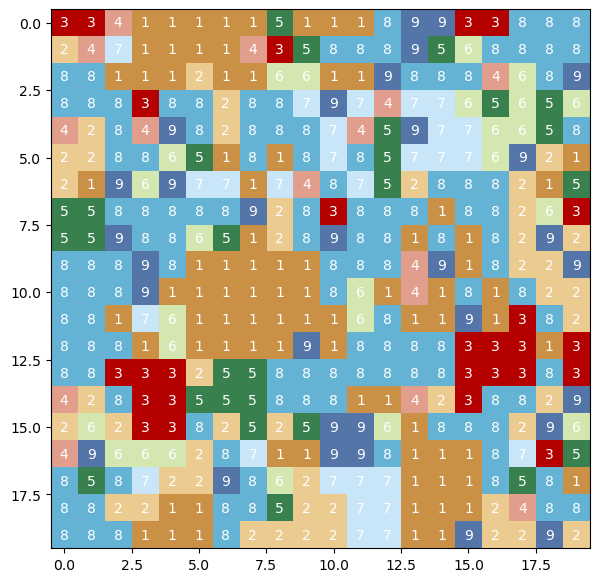
cost\_filter=['lu\_area\_cost']  
  
pattern\_generated\_random\_lu\_area,epoch=usda\_heuristicsw.genetic\_algorithm\_2d\_fixed\_map(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 population\_size=50,  
 generations=100,  
 mutation\_rate=0.5,  
 target\_function=target\_function\_jisperveld\_plan,  
 crossover\_name='crossover\_CBO',  
 mutation\_name='mutation\_MPO\_MBO',   
 mpo\_mbo='mpo',  
 verbose=20)  
  
fig, ax=plt.subplots()  
ax.plot(epoch.keys(),epoch.values())  
plt.show()

Generation = 0; f(x) = 18385554.0  
Generation = 20; f(x) = 10971234.0  
Generation = 40; f(x) = 2758514.0  
Generation = 60; f(x) = 122068.0  
Generation = 80; f(x) = 9089.0



unique, counts=np.unique(pattern\_generated\_random\_lu\_area, return\_counts=True)  
unique\_counts=dict(zip(unique, counts))  
print(unique\_counts)  
usda\_vis.imshow\_label2darray(pattern\_generated\_random\_lu\_area,figsize=(7,7),cmap=cmap\_LC,norm=norm,fontsize=10)

{1: 83, 2: 46, 3: 26, 4: 15, 5: 27, 6: 25, 7: 27, 8: 120, 9: 31}

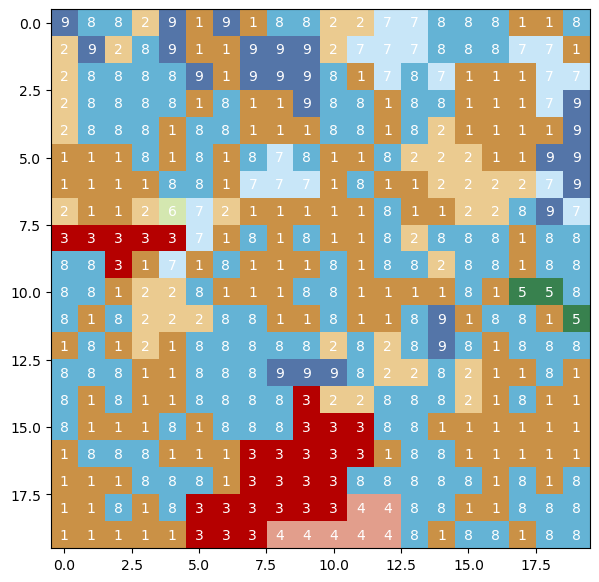


* 随机初始化种群的多目标空间规划

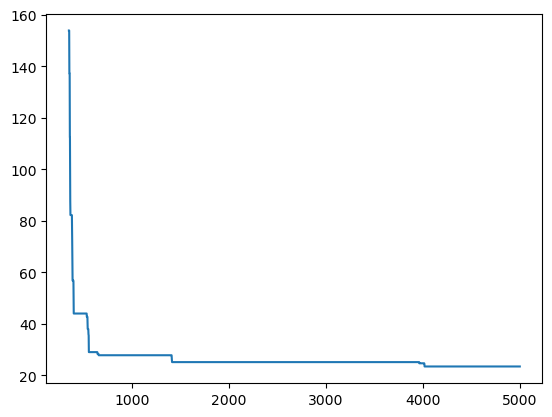
随机初始化种群，迭代5000次，从空间目标上判断，土地利用类型的破碎化程度减小，最大簇相对大小有所改善，紧密度增加，对应的值分别为38，6.185和0.347，与最初设置的目标值（取了最大和最小值的中间值，分别为27，5.4和0.5）接近；从准则目标上判断，自然价值、娱乐休闲价值和土地利用类型转换成本依次为2332、2054和-15840.0，而最初设置的目标值为2200、2200和-1940000.0，其中自然价值、娱乐休闲价值接近目标值，土地利用转换成本只要小于0就满足要求，因此准则目标也达到了最初设置的目标值；对于土地利用类型的数量空值，计算结果为{1: 135, 2: 38, 3: 28, 4: 7, 5: 3, 6: 1, 7: 21, 8: 143, 9: 24}，均位于最初指定的区间内。因此，可以判断应用遗传算法于多目标空间决策是可行的，其计算结果能够为空间规划提供参照。

cost\_filter=['natrue\_cost','recreation\_cost','lu\_conversion\_cost','C','L','R','lu\_area\_cost']  
  
pattern\_generated\_random,epoch=usda\_heuristicsw.genetic\_algorithm\_2d\_fixed\_map(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 population\_size=50,  
 generations=5001,  
 mutation\_rate=0.5,  
 target\_function=target\_function\_jisperveld\_plan,  
 crossover\_name='crossover\_CBO',  
 mutation\_name='mutation\_MPO\_MBO',   
 mpo\_mbo='mpo',  
 fixed\_map=fixed\_map,  
 verbose=500)  
  
usda\_vis.imshow\_label2darray(pattern\_generated\_random,figsize=(7,7),cmap=cmap\_LC,norm=norm,fontsize=10)

Generation = 0; f(x) = 1.5770013236205606e+22  
Generation = 500; f(x) = 43.99567511055381  
Generation = 1000; f(x) = 27.804721274539045  
Generation = 1500; f(x) = 25.13594265347618  
Generation = 2000; f(x) = 25.13594265347618  
Generation = 2500; f(x) = 25.13594265347618  
Generation = 3000; f(x) = 25.13594265347618  
Generation = 3500; f(x) = 25.13594265347618  
Generation = 4000; f(x) = 24.67886469436648  
Generation = 4500; f(x) = 23.440904167399868  
Generation = 5000; f(x) = 23.440904167399868



fig, ax=plt.subplots()  
ax.plot(list(epoch.keys())[350:],list(epoch.values())[350:])  
plt.show()



计算最优解的准则目标、空间目标和土地利用类型数量控制值。

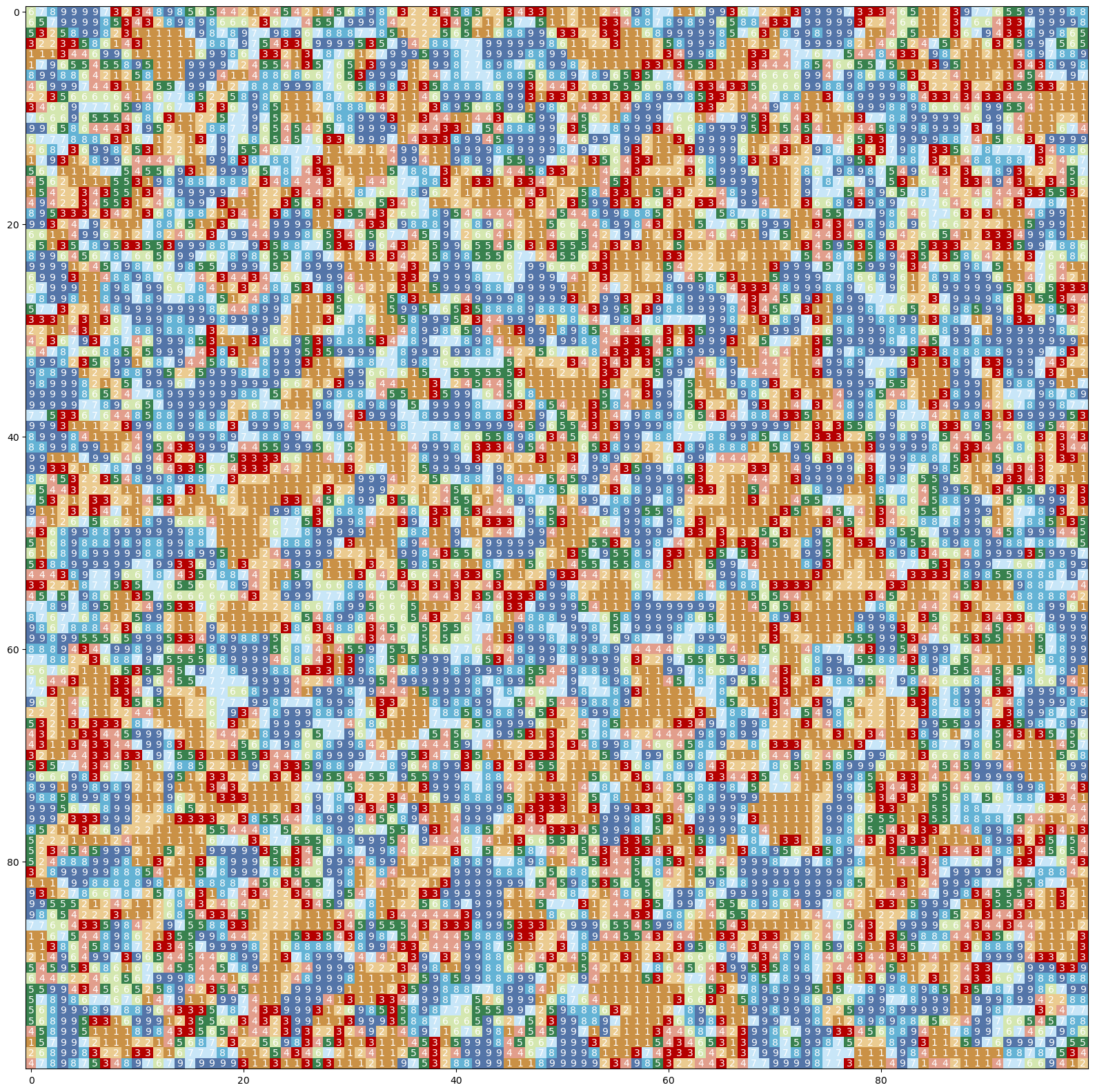
C,L,R=target\_function\_spatial\_objectives\_constrains(pattern\_generated\_random)   
print(C,L,R)  
  
nature\_cost,recreation\_cost,conversion\_cost=target\_function\_additive\_objectives\_constrains(pattern\_generated\_random)  
print(nature\_cost,recreation\_cost,conversion\_cost)  
  
unique, counts=np.unique(pattern\_generated\_random, return\_counts=True)  
unique\_counts=dict(zip(unique, counts))  
print(unique\_counts)

38 6.185158116737064 0.3478124999999978  
2332 2054 -15840.0  
{1: 135, 2: 38, 3: 28, 4: 7, 5: 3, 6: 1, 7: 21, 8: 143, 9: 24}

**D - 基于标记距离的2维度布局生成**

下面进行了一个单个空间目标的实验，以最小化到一个土地利用或土地覆盖样方类/簇大小直方图标记特征的距离为目标，通过2维度遗传算法，使生成的样方类/簇大小直方图标记特征趋近于已知的样方，并调大了样方的大小，由之前实验的 20×20 的栅格数据，扩大为 100×100，观察成本值收敛情况。从计算结果来看，最优解到已知样方的距离为0.0849，所用的Jensen-Shannon距离区间为[0,1]，当值趋近于0时，两个样方的类/簇大小直方图标记特征基本相同，同时，随着迭代次数的增加，成本值为缓慢下降趋势，因此表明2维度标记特征用于空间目标是可行的。

size=100  
X,\_=usda\_datasets.generate\_categorical\_2darray(size=size,seed=99)  
X4=mapclassify.FisherJenks(X[0], k=9).yb.reshape(size,size)+1  
usda\_vis.imshow\_label2darray(X4,figsize=(20,20),fontsize=10,cmap=cmap\_LC,norm=norm)



定义基于类/簇大小直方图标记特征距离的目标函数。

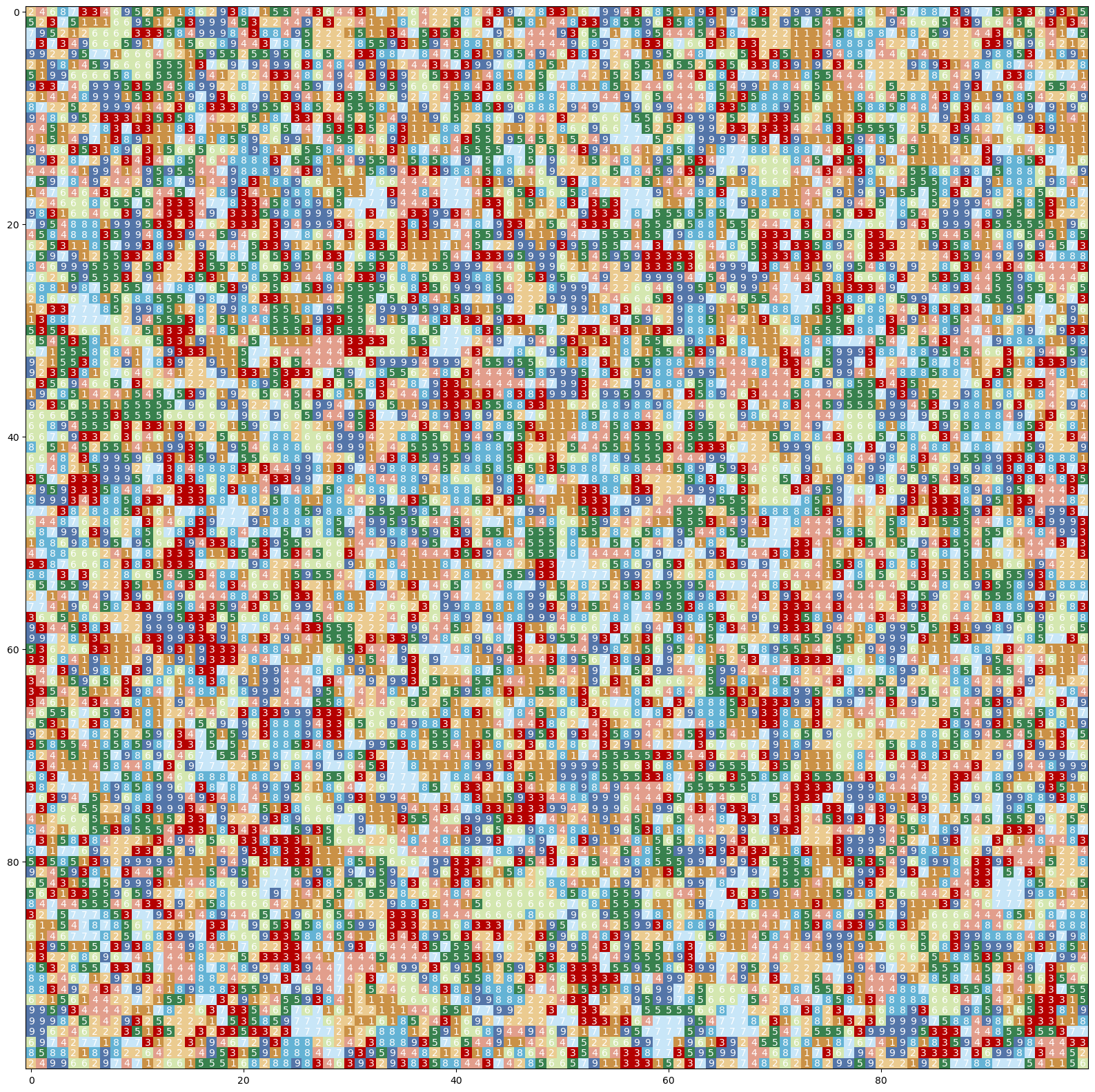
def target\_function\_class\_clumpSize(quadrat):  
 global compared\_quadradt   
  
 q1\_cc=cc3d.connected\_components(quadrat,connectivity=8,return\_N=False,out\_dtype=np.uint64)  
 q1\_cs=usda\_signature.class\_clumpSize\_histogram(quadrat,q1\_cc)   
   
 q2\_cc=cc3d.connected\_components(compared\_quadradt, connectivity=8,return\_N=False,out\_dtype=np.uint64)  
 q2\_cs=usda\_signature.class\_clumpSize\_histogram(compared\_quadradt,q2\_cc)   
   
 q1\_cs\_pdf=q1\_cs/q1\_cs.values.sum()  
 q2\_cs\_pdf=q2\_cs/q2\_cs.values.sum()  
  
 q1\_cs\_pdf,q2\_cs\_pdf=usda\_utils.complete\_dataframe\_rowcols([q1\_cs\_pdf,q2\_cs\_pdf])   
   
 class\_clumpSize\_pdf\_shannon=usda\_signature.Distances(q1\_cs\_pdf.to\_numpy().flatten(),q2\_cs\_pdf.to\_numpy().flatten())  
 distance=class\_clumpSize\_pdf\_shannon.shannon()['Jensen-Shan']  
  
 return distance

基于2维遗传算法求解。

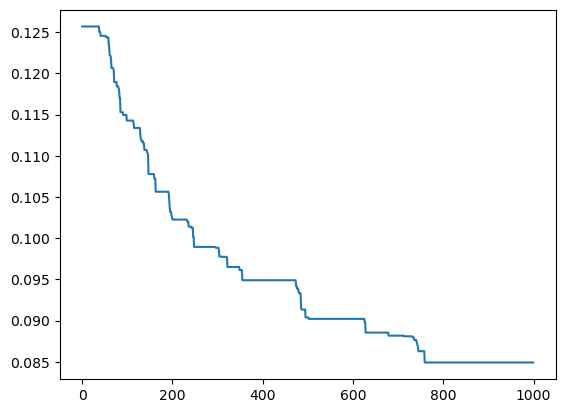
objects\_idx=list(range(1,10))  
rows\_n=size  
cols\_n=size  
compared\_quadradt=copy.deepcopy(X4)  
  
pattern\_generated,epoch=usda\_heuristicsw.genetic\_algorithm\_2d\_fixed\_map(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 population\_size=50,  
 generations=1000,  
 mutation\_rate=0.5,  
 target\_function=target\_function\_class\_clumpSize,  
 crossover\_name='crossover\_CBO',  
 mutation\_name='mutation\_MPO\_MBO',   
 mpo\_mbo='mpo',  
 verbose=200)

Generation = 0; f(x) = 0.1256898055950527  
Generation = 200; f(x) = 0.1022821600839174  
Generation = 400; f(x) = 0.09489259290469998  
Generation = 600; f(x) = 0.09020153855631115  
Generation = 800; f(x) = 0.08491062724665702

usda\_vis.imshow\_label2darray(pattern\_generated,figsize=(20,20),fontsize=10,cmap=cmap\_LC,norm=norm)



fig, ax=plt.subplots()  
ax.plot(epoch.keys(),epoch.values())  
plt.show()



## 3.3.2 元启发式算法（Meta-Heuristic Algorithm）

元启发式算法通常基于直观或经验构造的算法，给出解决优化问题（通常为最小化或最大化）的一个可行解，该可行解与最优解的偏离程度一般不能被预计。许多元启发式算法实现某种形式的随机优化，因此找到的解决方案取决于生成的随机变量集。在组合优化中，通过搜索大量可行的解决方案，元启发式算法通常可以用比优化算法（optimization algorithms）、迭代方法（ iterative methods）或简单启发式算法更少的计算量找到好的解决方案。在2000到2010年左右，及至今，受自然启发的优化算法大量出现，很多算法被正式非常有效，例如遗传算法（genetic algorithms）、粒子群优化算法（particle swarm optimization）、布谷鸟搜索（cuckoo search）和萤火虫算法（firefly algorithms ）等。这些算法可以应用到多准则决策（Multiple Criteria Decision-Making，MCDM），多目标空间规划中，能够为城市规划提供有价值的计算信息。 Valdecy Pereira 建构了[pyMetaheuristic](https://github.com/Valdecy/pyMetaheuristic)①库，集成了约40种元启发式算法，这为应用元启发式算法于规划领域提供了算法集成，具有重要价值。下述表格罗列了这些算法的名称和对应的参考文献，方便研究检索。

| 序号 | 算法名 | 参考论文 | 备注 |
| --- | --- | --- | --- |
| 1 | Adaptive Random Search | Adaptive random search methods for simulation optimization[14] |  |
| 2 | Ant Lion Optimizer | The Ant Lion Optimizer[15] |  |
| 3 | Arithmetic Optimization Algorithm | The Arithmetic Optimization Algorithm[16] |  |
| 4 | Artificial Bee Colony Optimization | An Idea Based on Honey Bee Swarm for Numerical Optimization[17] |  |
| 5 | Artificial Fish Swarm Algorithm | 一种基于动物自治体的寻优模式:鱼群算法[18] |  |
| 6 | Bat Algorithm | A New Metaheuristic Bat-Inspired Algorithm[19] |  |
| 7 | Biogeography Based Optimization | Biogeography-based optimization[20] |  |
| 8 | Cross-Entropy Method | Optimization of computer simulation models with rare events[21] |  |
| 9 | Crow Search Algorithm | A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm[22] |  |
| 10 | Cuckoo Search | Cuckoo Search via Levy Flights[23] |  |
| 11 | Differential Evolution | Differential Evolution-A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces[24] |  |
| 12 | Dispersive Flies Optimization | Dispersive flies optimisation and medical imaging[25] |  |
| 13 | Dragonfly Algorithm | Dragonfly algorithm: a comprehensive review and applications[26] |  |
| 14 | Firefly Algorithm | Nature-inspired optimization algorithms[27] |  |
| 15 | Flow Direction Algorithm | Flow Direction Algorithm (FDA): a Novel Optimizer Approach for Solving Optimization Problems[28] |  |
| 16 | Flower Pollination Algorithm | Nature-inspired optimization algorithms[27] |  |
| 17 | Genetic Algorithm | Adaptation in Natural and Artificial Systems\_ An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence[29] |  |
| 18 | Grey Wolf Optimizer | Grey Wolf Optimizer[30] |  |
| 19 | Grasshopper Optimization Algorithm | Grasshopper Optimisation Algorithm: Theory and application[31] |  |
| 20 | Gravitational Search Algorithm | GSA: A Gravitational Search Algorithm[32] |  |
| 21 | Harris Hawks Optimization | Harris hawks optimization: Algorithm and applications[33] |  |
| 22 | Improved Grey Wolf Optimizer | An improved grey wolf optimizer for solving engineering problems[34] |  |
| 23 | Improved Whale Optimization Algorithm | IWOA: An improved whale optimization algorithm for optimization problems[35] |  |
| 24 | Jaya | Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems[36] |  |
| 25 | Jellyfish Search Optimizer | A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean[37] |  |
| 26 | Krill Herd Algorithm | A comprehensive review: Krill Herd algorithm (KH) and its applications[38] |  |
| 27 | Memetic Algorithm | On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms[39] |  |
| 28 | Moth Flame Optimization | Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm[40] |  |
| 29 | Multiverse Optimizer | Multi-Verse Optimizer: a nature-inspired algorithm for global optimization[41] |  |
| 30 | Pathfinder Algorithm | A new meta-heuristic optimizer: Pathfinder algorithm[42] |  |
| 31 | Particle Swarm Optimization | Particle swarm optimization[43] |  |
| 32 | Random Search | Recent Advances in Finding Best Operating Conditions[44] |  |
| 33 | Salp Swarm Algorithm | Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems[45] |  |
| 34 | Simulated Annealing | Optimization by Simulated Annealing[46] |  |
| 35 | Sine Cosine Algorithm | A Sine Cosine Algorithm for solving optimization problems[47] |  |
| 36 | Student Psychology Based Optimization | Student psychology based optimization algorithm: A new population based optimization algorithm for solving optimization problems[48] |  |
| 37 | Symbiotic Organisms Search | Symbiotic Organisms Search: A new metaheuristic optimization algorithm[49] |  |
| 38 | Teaching Learning Based Optimization | Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems[50] |  |
| 39 | Whale Optimization Algorithm | The Whale Optimization Algorithm [51] |  |

对上述元启发式算法，在**权重决策**部分阐释了遗传算法和灰狼算法，前文应用遗算法对1维向二维的多目标空间规划途径做了阐释，下面从上述元启发式算法中选择了粒子群优化算法、布谷鸟搜索和萤火虫算法给出阐释，并就粒子群算法在空间2维度上进行了初步探索。

### 3.3.2.1 粒子群优化算法（Particle Swarm Optimization, PSO）

元启发式算法通常存在类似的表述，对于解（solution，为1维到多维的向量），遗传算法（GA）表述为染色体（chromosome），灰狼算法（GWO）表述为灰狼，粒子群（PSO）算法表述为粒子；对于多个解组成的集群，GA表述为种群（population），GWO表述为狼群位置（position（swarm）），PSO表述为粒子群位置（position（swarm））。GA初始化时仅需要初始化种群（随机多个解），PSO初始化时则初始化粒子群位置（随机多个解）同时，需要初始化对应粒子位置的速度向量。一个粒子的位置向量（解）表述为；对应的一个粒子速度向量为，式中，表示粒子的序号，表示求解问题的维数。求解过程中存在两个关键向量（解），一个是各个粒子自身的历史（局部）最优解（pBest）；另一个是全局（所有粒子）的最优解（gBest），每次迭代过程中各个粒子的位置向量更新计算公式为：，式中，部分为惯性（inertia），使得粒子沿原有相同方向和相同大小运动，发现有潜在最优解的区域，寻找新解。为惯性权重，为正的常数，通过调整其值的大小，用于平衡全局搜索（global search/exploration）（趋于高值）和局部搜索（local search/exploitation）（趋于低值）。部分为粒子自我认知（personal influence）部分，比较粒子历史最优解与当前解的距离。为一个正常数，为个体认知参数，

c2c1=c2=0c1>0,c2=0c2>0,c1=0c1=c2≠0时，所有粒子都被吸引到pBest和gBest的均值位置。

位置向量更新计算公式代码表述为：

# Function: Velocity  
def velocity\_vector(position, init\_velocity, i\_b\_matrix, best\_global, w = 0.5, c1 = 2, c2 = 2):  
 r1 = int.from\_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)  
 r2 = int.from\_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)  
 velocity = np.zeros((position.shape[0], init\_velocity.shape[1]))  
 for i in range(0, init\_velocity.shape[0]):  
 for j in range(0, init\_velocity.shape[1]):  
 velocity[i,j] = w\*init\_velocity[i,j] + c1\*r1\*(i\_b\_matrix[i,j] - position[i,j]) + c2\*r2\*(best\_global[j] - position[i,j])  
 return velocity  
  
# Function: Updtade Position  
def update\_position(position, velocity, min\_values = [-5,-5], max\_values = [5,5], target\_function = target\_function):  
 for i in range(0, position.shape[0]):  
 for j in range(0, position.shape[1] - 1):  
 position[i,j] = np.clip((position[i,j] + velocity[i,j]), min\_values[j], max\_values[j])  
 position[i,-1] = target\_function(position[i,0:position.shape[1]-1])  
 return position

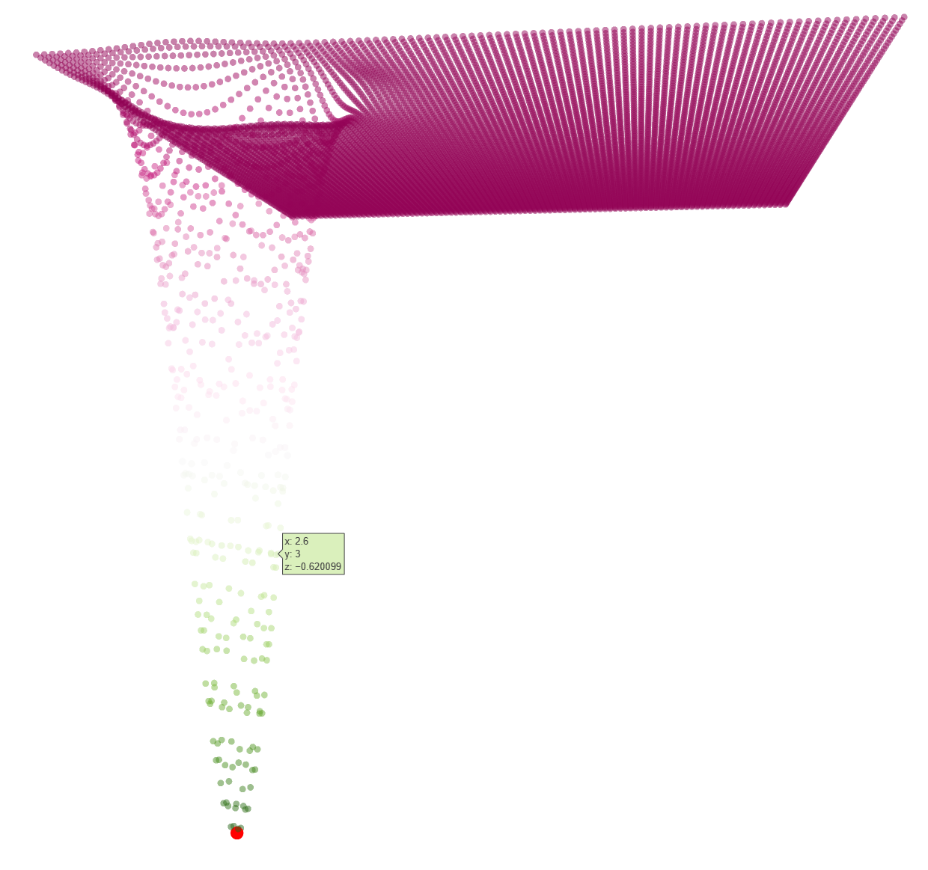
Valdecy Pereira 在测试 PSO 时选择了 Easom 函数：作为目标函数，计算结果如下。

def easom(variables\_values = [0, 0]):  
 x1, x2 = variables\_values  
 func\_value = -np.cos(x1) \* np.cos(x2) \* np.exp(-(x1 - np.pi) \*\* 2 - (x2 - np.pi) \*\* 2)  
 return func\_value  
  
parameters = {  
 'swarm\_size': 250,  
 'min\_values': (-5, -5),  
 'max\_values': (5, 5),  
 'iterations': 500,  
 'decay': 0,  
 'w': 0.9,  
 'c1': 2,  
 'c2': 2,  
 'verbose': 100  
}   
pso,epoch =usda\_heuristic.particle\_swarm\_optimization(target\_function = easom, \*\*parameters)  
# PSO - Solution  
variables = pso[:-1]  
minimum = pso[ -1]  
print('Variables: ', np.around(variables, 4) , ' Minimum Value Found: ', round(minimum, 4) )

Iteration = 0 f(x) = -0.6409483703488887  
Iteration = 100 f(x) = -0.9998786813759035  
Iteration = 200 f(x) = -0.999999903656637  
Iteration = 300 f(x) = -0.999999903656637  
Iteration = 400 f(x) = -0.999999903656637  
Iteration = 500 f(x) = -0.999999903656637  
Variables: [3.1418 3.1418] Minimum Value Found: -1.0

# PSO - Plot Solution  
plot\_parameters = {  
 'min\_values': (-5, -5),  
 'max\_values': (5, 5),  
 'step': (0.1, 0.1),  
 'solution': [variables],  
 'proj\_view': '3D',  
 'view': 'browser' # 'notebook'  
}  
usda\_vis.plot\_single\_function(target\_function = easom, \*\*plot\_parameters)

通过绘制函数2维分布，可以发现最优解的位置，位于最低端红点处。



* 基准函数（benchmark functions）

Mirjalili, S. [30] 总结了用于测试元启发式算法的基准函数，将其归为4组，为单峰函数（unimodal）、多峰函数（multimodal）、固定维度多峰函数（ﬁxed-dimension multimodal）和复合函数（composite functions）。下述引用表格[30]中，给出了函数（function），基准函数的维数（dimension）、函数搜索空间的边界（range）和最优值（optimum）。

单峰基准函数

| 函数 | 维数 | 边界 | 最优值 |
| --- | --- | --- | --- |
|  | 30 | [-100,100] | 0 |
|  | 30 | [-10,10] | 0 |
|  | 30 | [-100,100] | 0 |
|  | 30 | [-100,100] | 0 |
|  | 30 | [-30,30] | 0 |
|  | 30 | [-100,100] | 0 |
|  | 30 | [-1.28,1.28] | 0 |

多峰基准函数

| 函数 | 维数 | 边界 | 最优值 |
| --- | --- | --- | --- |
|  | 30 | [-500,500] | -418.9829×5 |
|  | 30 | [-5.12,5.12] | 0 |
|  | 30 | [-32,32] | 0 |
|  | 30 | [-600,600] | 0 |
|  | 30 | [-50,50] | 0 |
|  | 30 | [-50,50] | 0 |
|  | 30 | [0,$ $] | -4.687 |
|  | 30 | [-20,20] | -1 |
|  | 30 | [-10,10] | -1 |

固定维度多峰基准函数

| 函数 | 维数 | 边界 | 最优值 |
| --- | --- | --- | --- |
|  | 2 | [-65,65] | 1 |
|  | 4 | [-5,5] | 0.00030 |
|  | 2 | [-5,5] | -1.0316 |
|  | 2 | [-5,5] | 0.398 |
|  | 2 | [-2,2] | 3 |
|  | 3 | [1,3] | -3.86 |
|  | 6 | [0,1] | -3.32 |
|  | 4 | [0,10] | -10.1532 |
|  | 4 | [0,10] | -10.4028 |
|  | 4 | [0,10] | -10.5363 |

复合基准函数

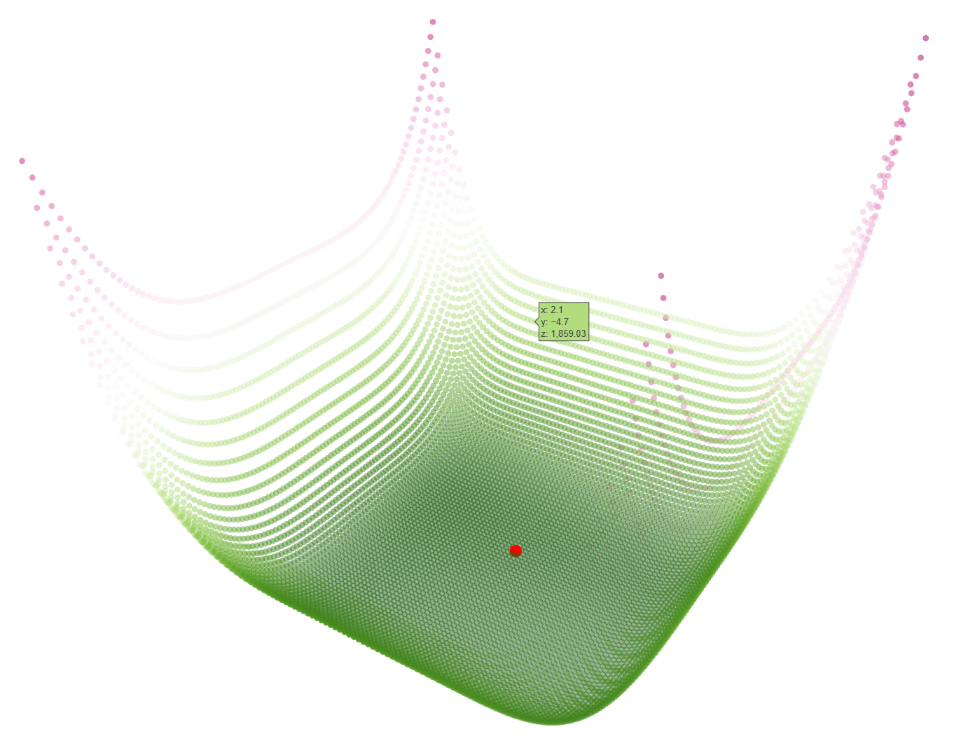
| 函数 | 维数 | 边界 | 最优值 |
| --- | --- | --- | --- |
|  | 10 | [-5,5] | 0 |
|  | 10 | [-5,5] | 0 |
|  | 10 | [-5,5] | 0 |
|  | 10 | [-5,5] | 0 |
|  | 10 | [-5,5] | 0 |
|  | 10 | [-5,5] | 0 |

从上面选择了一个基础的基准函数，再次测试 PSO， 其计算结果为 -1.0316，这与上述给出的最优解同。

def f\_16(variables\_values = [0, 0]):  
 x1, x2=variables\_values  
 func\_value=4\*pow(x1,2)-2.1\* pow(x1,4)+1/3\*pow(x1,6)+x1\*x2-4\*pow(x2,2)+4\*pow(x2,4)  
   
 return func\_value  
  
parameters = {  
 'swarm\_size': 250,  
 'min\_values': (-5, -5),  
 'max\_values': (5, 5),  
 'iterations': 500,  
 'decay': 0,  
 'w': 0.9,  
 'c1': 2,  
 'c2': 2,  
 'verbose': 100  
}   
pso\_f16,epoch=usda\_heuristic.particle\_swarm\_optimization(target\_function = f\_16, \*\*parameters)  
# PSO - Solution  
variables = pso\_f16[:-1]  
minimum = pso\_f16[ -1]  
print('Variables: ', np.around(variables, 4) , ' Minimum Value Found: ', round(minimum, 4) )

Iteration = 0 f(x) = -0.9384786598518445  
Iteration = 100 f(x) = -1.0313622604609696  
Iteration = 200 f(x) = -1.0315996120260253  
Iteration = 300 f(x) = -1.03160271424224  
Iteration = 400 f(x) = -1.0316119571982225  
Iteration = 500 f(x) = -1.0316119571982225  
Variables: [ 0.0878 -0.7128] Minimum Value Found: -1.0316

plot\_parameters = {  
 'min\_values': (-5, -5),  
 'max\_values': (5, 5),  
 'step': (0.1, 0.1),  
 'solution': [variables],  
 'proj\_view': '3D',  
 'view': 'browser' # 'notebook'  
}  
usda\_vis.plot\_single\_function(target\_function = f\_16, \*\*plot\_parameters)



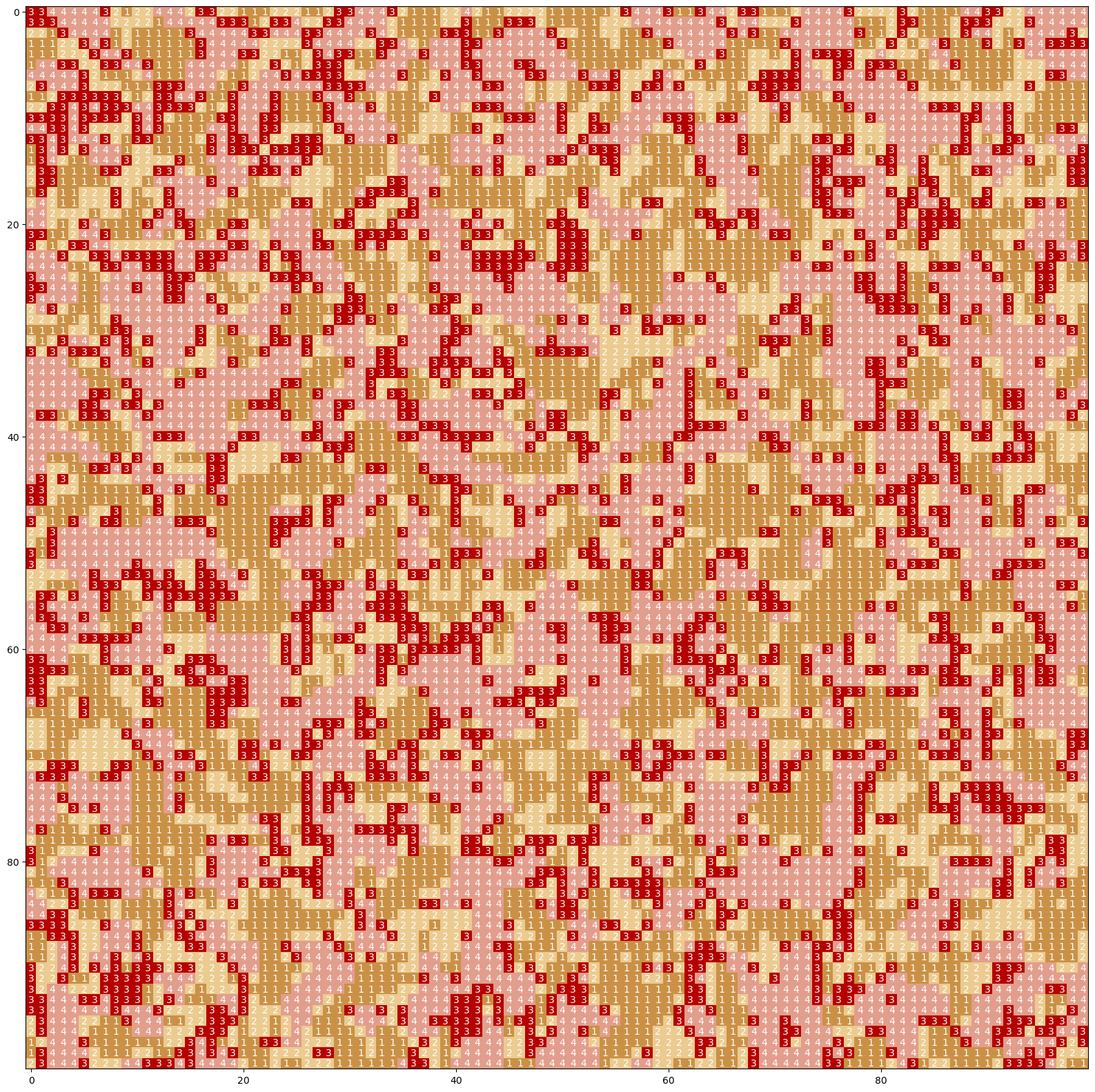
* PSO 2维度思考

GA 从1维度到2维度，关键是染色体（解）维度的变化，及对2维度染色体交叉和变异方式的调整。PSO 在1维度上时一个粒子的位置向量和速度向量是一一对应的关系，根据全局最优位置和历史最优位置更新粒子的速度向量。当依据 PSO 的基本思想，将其转换到2维度空间用于单目标和多目标非线性空间规划时，位置向量变为2维的矩阵，值为土地利用（土地覆盖）的分类值，即将分类值看作粒子所在的位置，位置的更新就是分类值的变化，例如粒子由农田转换为林地等。那么对于粒子位置更新所依据的速度向量，转换为位置更新的概率，概率分布为粒子（当前位置，即占有的一类土地利用类型）转换到所有各个土地利用类型的概率，表示为，对应有个值，为土地利用类型的数量，土地利用类型索引用表示，有。概率值表述为，位于[0,1]之间。行为和列为粒子的概率值表述为，且满足，即一个粒子的概率和为1。依据 PSO 对gBest和pBest的定义，对于2维粒子也受到和的影响，例如当前粒子的土地利用类型为农田，gBest对应位置的粒子为林地，而pBest对应位置的粒子为水体，即该粒子有向林地和水体转换的可能。同时，PSO 有一个惯性部分，用于保持粒子沿原有方向和速度运动，在2维中，表述为粒子保持自身所持有土地类型不变的可能性。因此，因此考虑到粒子的惯性部分，自我认知和社会经验部分，沿用惯性权重，用于调整粒子保持自身土地类型不变的能力；将学习参数和合并为一个，称为平衡参数，值位于[0,1]之间；保留两个随机数和，增加一定的随机性。因此有，对粒子的影响程度为，对粒子的影响程度为。保持粒子概率中对应和的概率值不变，同时保持粒子自身的概率值为不变，其中为例子自身索引值。将不在[,,]中的其它概率值同作为一个待重新分配的“概率池”，按照全局和历史最优位置的影响程度分配到各自已有概率上。因为有的影响，剩余的部分重新按照已有概率的比例分配回原来对应的土地利用类型上，从而更新了已有的概率分布，即速度向量。

因为空间规划中，同一类型的土地利用类型通常会集聚在一起，而原有的 PSO 方法中仅考虑粒子自身在全局和历史最优位置的影响，不涉及邻近粒子的影响，因此为了避免解的“椒盐”现象，计算当前位置的粒子到邻里8个位置粒子的频数，如果土地利用类型不在邻里单元中，则配置概率值为0，并标准化到[0,1]之间，使其和为1，公式为： frckFrcwnghPrcFrc。

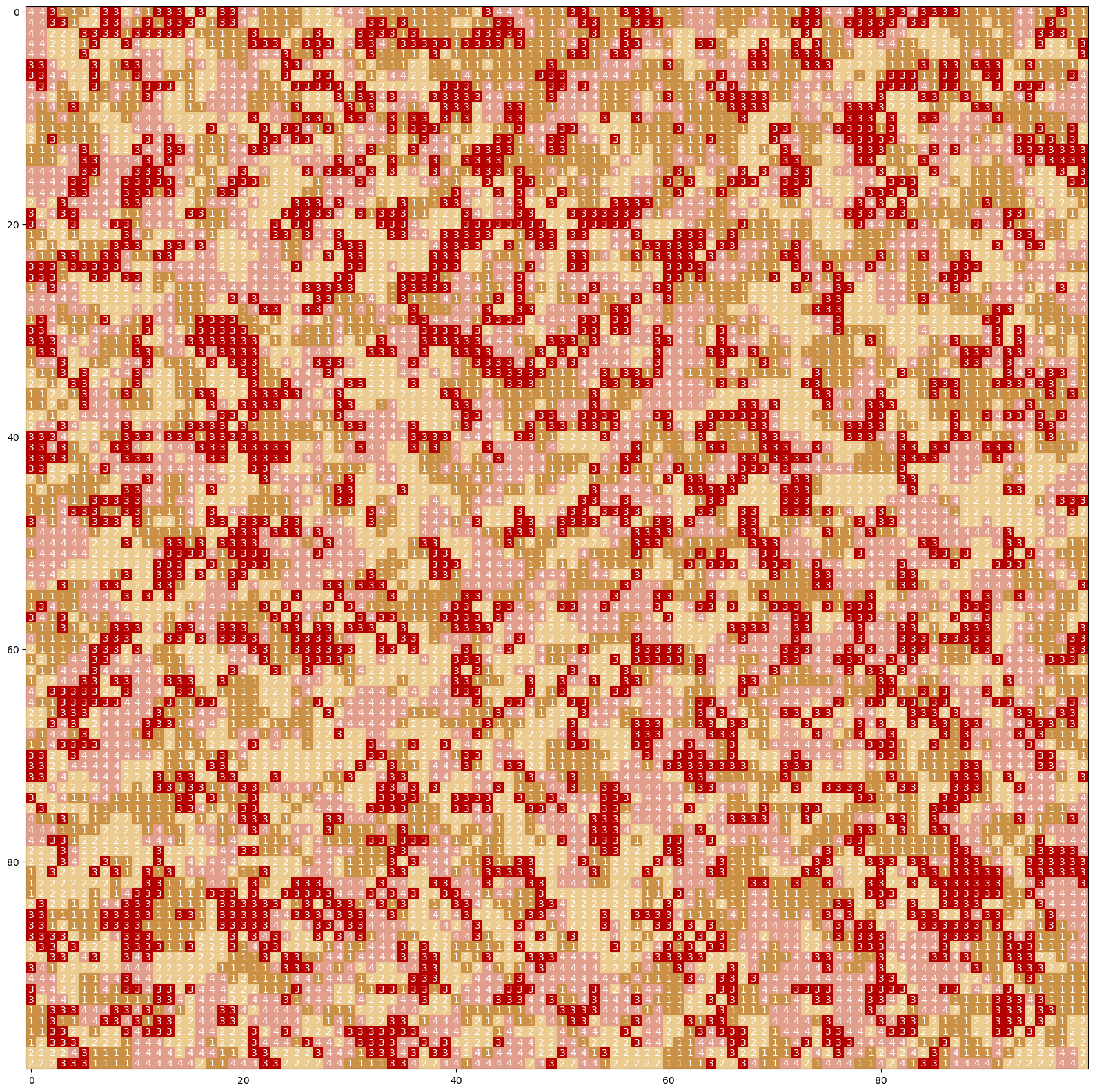
使用2维 PSO 算法计算类/簇大小直方图标记特征距离为目标的最优解，配置了土地利用类型为4类，配置ngh\_w参数分别为0.2和0.5，从计算结果来看，成本值曲线快速收敛，10次迭代的最终成本值结果分别为0.0668和0.074，但可以看到当调大ngh\_w后，单个簇的大小有所增加。同时也进行了9类分类的计算，同样可以达到0.07左右的水平上，但是单个簇的大小较之4个分类要小，在进一步的探索中可以尝试，1，增加空间目标约束条件；2， 调整邻里粒子速度向量由邻里8个邻接单元，扩大到以目标粒子为中小更大的范围。

size=100  
X,\_=usda\_datasets.generate\_categorical\_2darray(size=size,seed=99)  
X4=mapclassify.FisherJenks(X[0], k=4).yb.reshape(size,size)+1  
usda\_vis.imshow\_label2darray(X4,figsize=(20,20),fontsize=10,cmap=cmap\_LC,norm=norm)



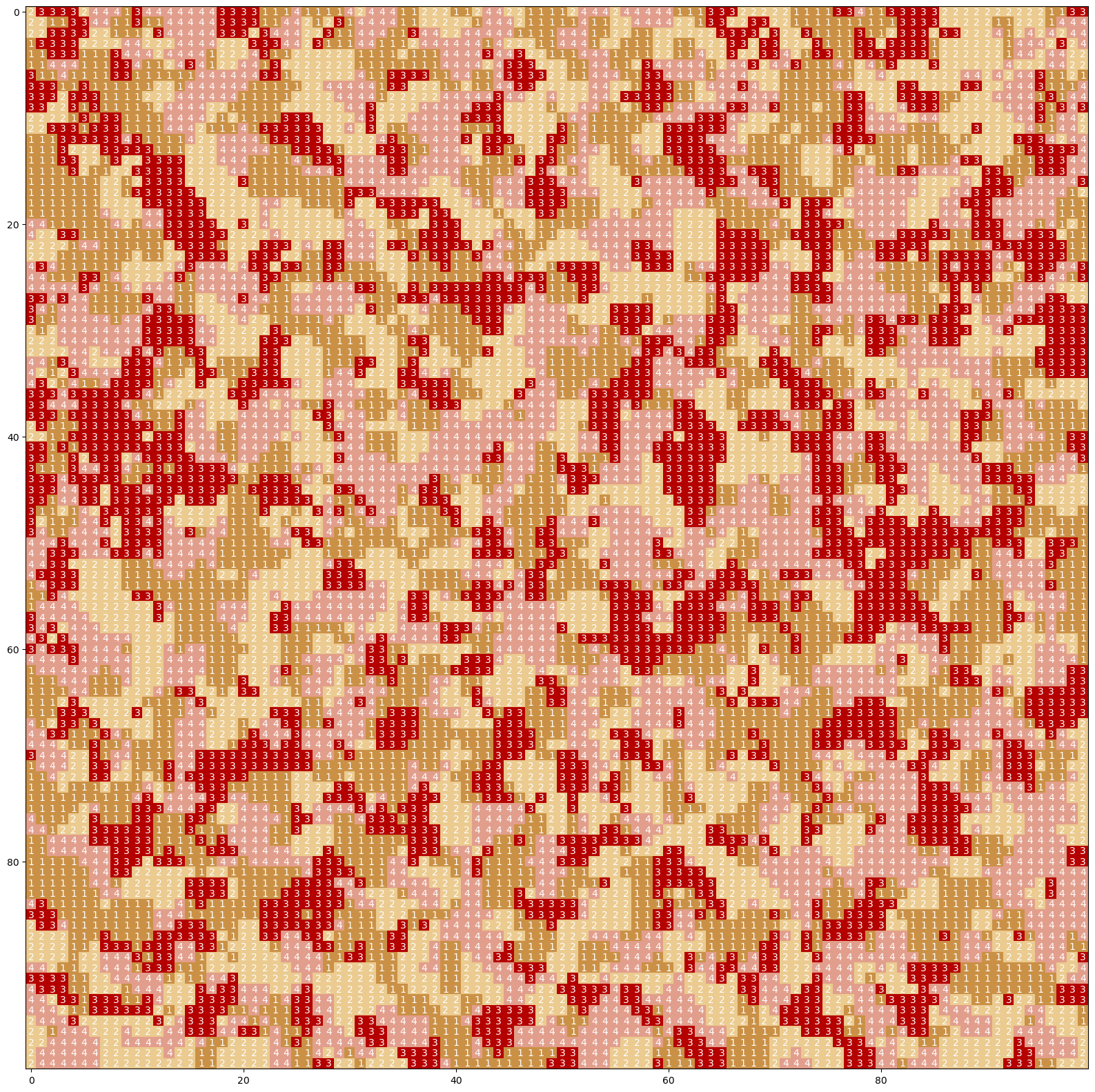
objects\_idx=list(range(1,5))  
rows\_n=size  
cols\_n=size  
compared\_quadradt=copy.deepcopy(X4)  
  
pattern\_generated\_pso,epoch=usda\_heuristic.particle\_swarm\_optimization\_2d(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 c=0.5,  
 w=0.9,  
 ngh\_w=0.2,   
 swarm\_size=20,  
 target\_function=target\_function\_class\_clumpSize,   
 iterations=10,  
 verbose=2)   
  
usda\_vis.imshow\_label2darray(pattern\_generated\_pso,figsize=(20,20),fontsize=10,cmap=cmap\_LC,norm=norm)

Iteration = 0 f(x) = 0.15734583591594925  
Iteration = 2 f(x) = 0.10394128255689686  
Iteration = 4 f(x) = 0.07785195388011691  
Iteration = 6 f(x) = 0.0774580960602753  
Iteration = 8 f(x) = 0.07242626375591905  
Iteration = 10 f(x) = 0.066886064219671



pattern\_generated\_pso,epoch=usda\_heuristic.particle\_swarm\_optimization\_2d(  
 objects\_idx,  
 rows\_n=rows\_n,  
 cols\_n=cols\_n,  
 c=0.5,  
 w=0.9,  
 ngh\_w=0.5,   
 swarm\_size=20,  
 target\_function=target\_function\_class\_clumpSize,   
 iterations=10,  
 verbose=2)   
  
usda\_vis.imshow\_label2darray(pattern\_generated\_pso,figsize=(20,20),fontsize=10,cmap=cmap\_LC,norm=norm)

Iteration = 0 f(x) = 0.1532896680592426  
Iteration = 2 f(x) = 0.07797312494712563  
Iteration = 4 f(x) = 0.07429198750896798  
Iteration = 6 f(x) = 0.07429198750896798  
Iteration = 8 f(x) = 0.07429198750896798  
Iteration = 10 f(x) = 0.07429198750896798



### 3.3.2.2 布谷鸟搜索算法（Cuckoo Search, CS）

* CS 仿生物智能优化算法，根据布谷鸟孵育寄生的行为特征构建，作者制定了3个理想化的规则如下：

1. 每只布谷鸟一次产一个蛋，并将其蛋倾倒在随机选择的巢中；
2. 产有高质量蛋的最好巢穴将传给下一代；
3. 可用宿主巢的数量是固定的，布谷鸟下的蛋被宿主鸟发现的概率为。在这种情况下，寄主鸟要么将蛋扔掉，要么弃巢而建一个全新的巢穴。

为简单起见约定，巢中的每个鸡蛋代表一个解决方案，布谷鸟蛋代表一个新的解决方案，目的是使用新的和可能更好的解决方案（布谷鸟）来替换一个巢中不太好的解决方案。

* 根据上述3条规则，CS 的基本步骤可以概括为如下伪代码：

Algorithm 1 Cuckoo Search

Objective function f(x),x=(x1​,…,xd​)T

Generate initial population of n host nests xi​(i=1,2,…,n)

while (i< MaxGeneration) or (( stop criterion) do

Get a cuckoo randomly by Lévy flights evaluate its quality/fitness Fi​

Choose a nest among  (n( say, j) randomly

if  (Fi​>Fj​) then

replace j by the new solution;

end if

A fraction  (pa​) of worse nests are abandoned and new ones are built;

Keep the best solutions (or nests with quality solutions);

Rank the solutions and find the current best

end while

Postprocess results and visualization

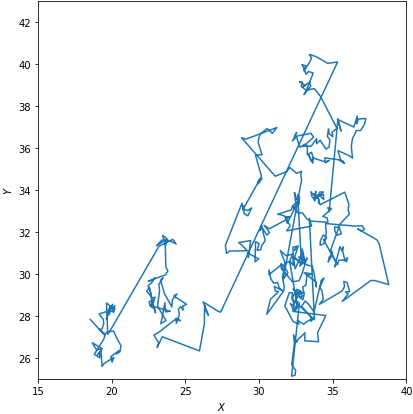
* 关键变量变化公式——全局Lévy飞行（全局搜索）和局部随机游走（局部搜索）

对于规则1（全局搜索），最明显的特点是应用了Lévy（莱维）飞行（一种随机游走）优化搜索，相比布朗运动（Brownian motion），Lévy飞行可以产生较大的跳跃。Lévy飞行是随机游走的步长服从Lévy分布的一种情况，可以由概率密度函数（Probability density function，PDF）或累积分布函数（cumulative distribution function，CDF）定义，对于 CDF，定义为：

， 式中，为步长；为最小步长；为幂律的斜率（slope of power law）。对于 PDF 定义为：

。如果调小将导致小步长的概率增加。对于Lévy飞行的直观理解，可以应用[mistree](https://github.com/knaidoo29/mistree)②库，打印Lévy飞行轨迹。从下述结果可以观察到Lévy飞行会产生较大的跳跃，这样就避免了过早的收敛导致为局部最优解，而能够在全局搜索，收敛到全局最优。

import numpy as np  
import matplotlib.pylab as plt  
import mistree as mist  
  
size=500  
x, y=mist.get\_levy\_flight(size, mode='2D')  
  
plt.figure(figsize=(6., 6.))  
plt.plot(x, y, '-', markersize=2., alpha=1)  
plt.xlabel(r'$X$')  
plt.ylabel(r'$Y$')  
plt.xlim(15., 40.)  
plt.ylim(25., 43.)  
plt.tight\_layout()  
plt.show()

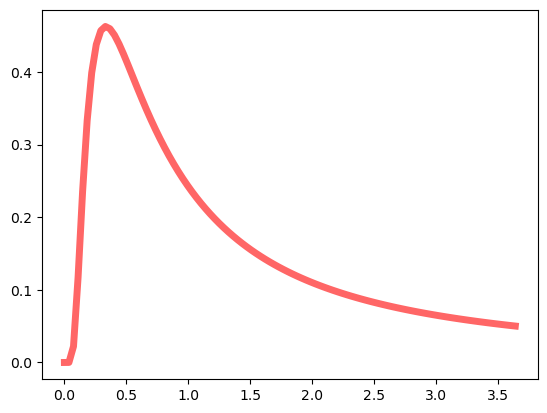


当产生一个新解（即第个布谷鸟）x(t+1) ，对应Lévy飞行和计算新解（含规则1和2）的代码为，

# Function: Levy Distribution  
def levy\_flight(mean):  
 x1 = math.sin((mean - 1.0)\*(random.uniform(-0.5\*math.pi, 0.5\*math.pi)) )/(math.pow(math.cos((random.uniform(-0.5\*math.pi, 0.5\*math.pi))), (1.0/(mean - 1.0))))  
 x2 = math.pow((math.cos((2.0 - mean)\*(random.uniform(-0.5\*math.pi, 0.5\*math.pi)))/(-math.log(random.uniform(0.0, 1.0)))), ((2.0 - mean)/(mean - 1.0)))  
 return x1\*x2  
  
# Function: Replace Bird  
def replace\_bird(position, alpha\_value = 0.01, lambda\_value = 1.5, min\_values = [-5,-5], max\_values = [5,5], target\_function = target\_function):  
 random\_bird = np.random.randint(position.shape[0], size = 1)[0]  
 new\_solution = np.zeros((1, position.shape[1]))   
 for j in range(0, position.shape[1] - 1):  
 new\_solution[0, j] = np.clip(position[random\_bird, j] + alpha\_value\*levy\_flight(lambda\_value)\*position[random\_bird, j]\*(int.from\_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)), min\_values[j], max\_values[j])  
 new\_solution[0,-1] = target\_function(new\_solution[0,0:new\_solution.shape[1]-1])  
  
 if (position[random\_bird, -1] > new\_solution[0,-1]):  
 position[random\_bird, :] = np.copy(new\_solution[0, :])  
 return position

上述公式中，为求解问题尺度相关的步长，多数情况下配置为值。Lévy飞行是步长的概率分布为重尾分布的随机行走，即在行走过程中有相对较高的概率出现大阔步，具有无限的均值和方差，这使得随机步长在搜索空间方面非常有效。SciPy库的scipy.stats.levy方法提供了Lévy连续随机变量计算，Lévy的 PDF 公式为：，式中。从下述代码打印的Lévy 的 PDF 可以观察到重尾分布的情形。

import numpy as np  
from scipy.stats import levy  
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots(1, 1)  
mean, var, skew, kurt = levy.stats(moments='mvsk')  
a, b = levy.ppf(0), levy.ppf(0.6)  
x = np.linspace(a, b, 100)  
ax.plot(x, levy.pdf(x),'r-', lw=5, alpha=0.6, label='levy pdf')  
plt.show()



对于规则3（局部搜索），计算方法为任选两个解求其差并乘以一个随机数（即步长）后与当前解求和，公式为：，式中，和为通过随机排列随机选择的两个不同解；是赫维赛德函数（ Heaviside function），是从均匀分布中抽取的随机数，是步长，表示两个向量的逐项乘积。对应代码为：

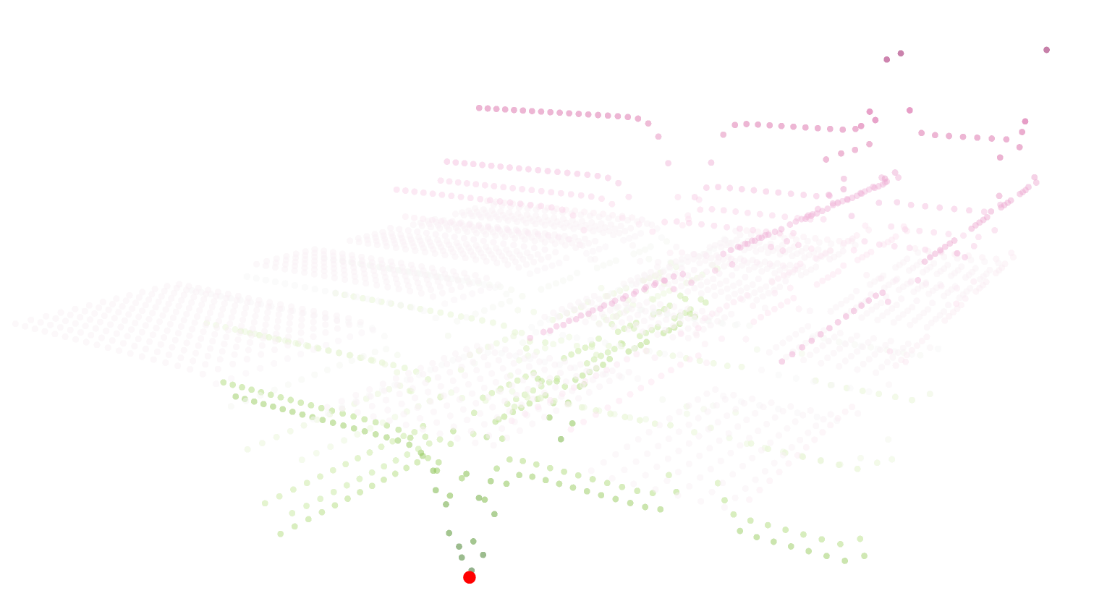
# Function: Update Positions  
def update\_positions(position, discovery\_rate = 0.25, min\_values = [-5,-5], max\_values = [5,5], target\_function = target\_function):  
 updated\_position = np.copy(position)  
 abandoned\_nests = math.ceil(discovery\_rate\*updated\_position.shape[0]) + 1  
 random\_bird\_j = np.random.randint(position.shape[0], size = 1)[0]  
 random\_bird\_k = np.random.randint(position.shape[0], size = 1)[0]  
 while(random\_bird\_j == random\_bird\_k):  
 random\_bird\_j = np.random.randint(position.shape[0], size = 1)[0]  
 nest\_list = list(position.argsort()[-(abandoned\_nests-1):][::-1][0])  
 for i in range(0, updated\_position.shape[0]):  
 for j in range(0, len(nest\_list)):  
 rand = int.from\_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)  
 if(i == nest\_list[j] and rand > discovery\_rate):  
 for k in range(0, updated\_position.shape[1] - 1):  
 rand = int.from\_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)  
 updated\_position[i,k] = np.clip(updated\_position[i,k] + rand\*(updated\_position[random\_bird\_j,k] - updated\_position[random\_bird\_k,k]), min\_values[k], max\_values[k])  
 updated\_position[i,-1] = target\_function(updated\_position[i,0:updated\_position.shape[1]-1])   
 return updated\_position

下面验证了作者在论文中使用的双变量Michalewicz函数，公式为：，式中，且，当解为时，有$f\_\* $。下述计算结果与上述基本吻合。

def michaelwicz(variables\_values=[0, 0]):  
 x, y=variables\_values  
 func\_value=-np.sin(x)\* pow(np.sin(x\*\*2/np.pi),20)-np.sin(y)\*pow(np.sin(2\*y\*\*2/np.pi),20)  
 return func\_value  
  
# CS - Parameters  
parameters = {  
 'birds': 50,  
 'min\_values': (0, 0),  
 'max\_values': (5, 5),  
 'iterations': 100,  
 'discovery\_rate': 0.25,  
 'alpha\_value': 0.01,  
 'lambda\_value': 1.5,  
 'verbose': 20  
}   
   
# CS - Algorithm  
cs,epoch=usda\_heuristic.cuckoo\_search(target\_function=michaelwicz, \*\*parameters)   
  
# CS - Solution  
variables=cs[:-1]  
minimum=cs[ -1]  
print('Variables: ', np.around(variables, 4) , ' Minimum Value Found: ', round(minimum, 4) )

Iteration = 0 f(x) = -1.3783857961267445  
Iteration = 20 f(x) = -1.7591905230703284  
Iteration = 40 f(x) = -1.8009891070897415  
Iteration = 60 f(x) = -1.801135422282869  
Iteration = 80 f(x) = -1.8012999441363957  
Iteration = 100 f(x) = -1.8012999441363957  
Variables: [2.2027 1.5705] Minimum Value Found: -1.8013

# CS - Plot Solution  
plot\_parameters = {  
 'min\_values': (0, 0),  
 'max\_values': (5, 5),  
 'step': (0.1, 0.1),  
 'solution': [variables],  
 'proj\_view': '3D',  
 'view': 'browser'   
}  
usda\_vis.plot\_single\_function(target\_function=michaelwicz, \*\*plot\_parameters)



### 3.3.2.3 萤火虫算法（Firefly Algorithm, FA）

* 理想化萤火虫的一些闪烁特征，确定3个理想化规则为：

1. 一只萤火虫会被其它萤火虫吸引，且不分性别；
2. 吸引力（Attractiveness）与萤火虫的光强度（Light Intensity）成正比。因此对于任何两只闪烁的萤火虫，较暗的萤火虫会向较亮的萤火虫移动，且吸引力随着距离的增加而减小。如果没有比特定萤火虫更亮的萤火虫，则其随机移动；
3. 萤火虫的亮度受目标函数的影响或决定。

对于最大化问题，亮度可以简单的与目标函数的值成比例。其它形式的亮度可以用与遗传算法中适应度函数（ﬁtness function）类似的方式定义。

* 根据上述3条规则，FA 的基本步骤可以概括为如下伪代码：

**Algorithm 2**Firefly Algorithm

Objective function *f*(***x***),***x***=(*x*1​,…,*xd*​)*T*.

Generate an initial population of *n* fireflies ***x****i*​(*i*=1,2,…,*n*).

Light intensity *Ii*​ at ***x****i*​ is determined by*f*(***x****i*​).

Define light absorption coefficient *γ*.

**while** (*t*< MaxGeneration)**do**

**for***i*=1:*n* (all *n* fireflies)**do**

**for***j*=1:*n* (all *n* fireflies) (inner loop)**do**

**if** (*Ii*​<*Ij*​)**then**

Move firefly *i* towards *j*.

**end if**

Vary attractiveness with distance*r* via exp[−*γr*2].

Evaluate new solutions and update light intensity.

**end for**

**end for**

Rank the fireflies and find the current global best ***g***∗​.

**end while**

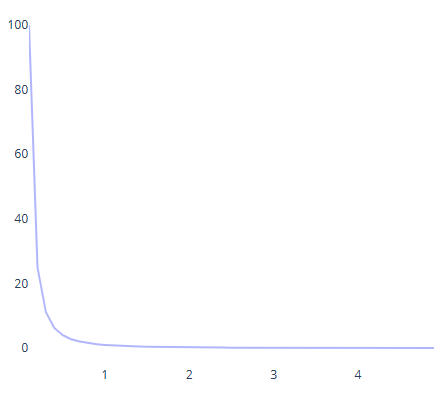
Postprocess results and visualization.

* 关键变量变化公式——光强度和吸引力

萤火虫的吸引力由其光强度决定，光强度与目标函数有关，因此最大优化问题的最简单情况是特定位置处萤火虫的光强度可以为：ijrij

I(r)为萤火虫的当前光强度。

def func(variables\_values=[0]):  
 r=variables\_values[0]   
 func\_val=1/pow(r,2)  
 return func\_val  
plot\_parameters={'min\_values': [0.1],'max\_values': [5],'step': [0.1], 'solution': [],'proj\_view': '1d','view': 'browser' , 'mode':'lines'}  
usda\_vis.plot\_single\_function(target\_function=func, \*\*plot\_parameters)

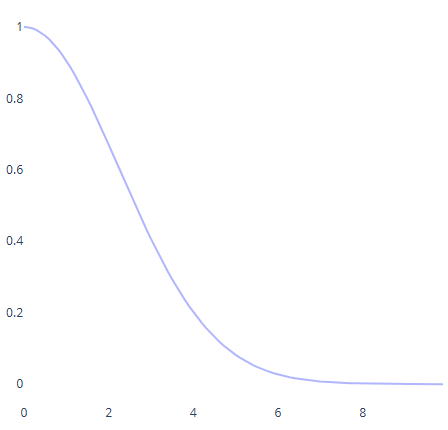


从上述打印的平方反比率曲线可以发现，平方反比率随距离的增加值迅速的下降，为了缓和曲线的下降速度，避免距离值不能为0的情况；且趋近于0时，值为正无穷，并增加光被介质吸收的影响，增加固定光吸收系数，用高斯（Gaussian）形式近似平方反比率和吸收的组合效应，公式为：，式中， I0为距离值为0时，当前萤火虫光强度。从下述打印的曲线可以观察到，距离为零时值为1，且曲线下降速度得以缓和。

def func(variables\_values=[0]):  
 r=variables\_values[0]   
 func\_val=pow(math.e,-1\*r\*\*2)  
 return func\_val  
plot\_parameters={'min\_values': [0],'max\_values': [5],'step': [0.1], 'solution': [],'proj\_view': '1d','view': 'browser' , 'mode':'lines'}  
usda\_vis.plot\_single\_function(target\_function=func, \*\*plot\_parameters)

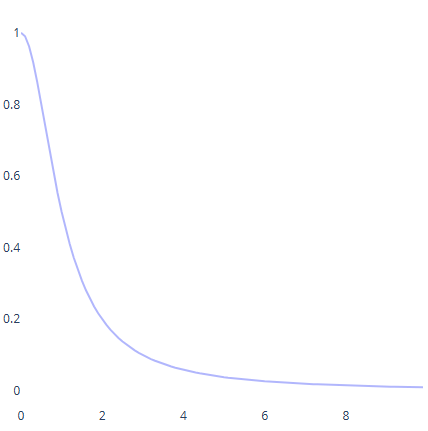
上述配置的值为1，当距离大于2时，值趋近于0；当调小该值为0.1后，从下述打印的曲线可以观察到，曲线下降的速度又进一步下降，当距离值高于6时，值趋近于0。

def func(variables\_values=[0]):  
 r=variables\_values[0]   
 func\_val=pow(math.e,-0.1\*r\*\*2)  
 return func\_val  
plot\_parameters={'min\_values': [0],'max\_values': [10],'step': [0.1], 'solution': [],'proj\_view': '1d','view': 'browser' , 'mode':'lines'}  
usda\_vis.plot\_single\_function(target\_function=func, \*\*plot\_parameters)



因为萤火中吸引力值与其光强度成正比，因此定义吸引力为：，式中，为距离值时当前萤火虫的吸引力。由于计算比计算指数函数快，因此如果需要，可以将上述公式近似为：。从下述打印的曲线，对比高斯形式，可以观察到二者近似的情况。吸引力函数定义了一个特征距离，，对于高斯形式吸引力显著的变化于到，对于近似形式则到。

def func(variables\_values=[0]):  
 r=variables\_values[0]   
 func\_val=1/(1+1\*r\*\*2)  
 return func\_val  
plot\_parameters={'min\_values': [0],'max\_values': [10],'step': [0.1], 'solution': [],'proj\_view': '1d','view': 'browser' , 'mode':'lines'}  
usda\_vis.plot\_single\_function(target\_function=func, \*\*plot\_parameters)



在实现中，吸引力可以是任何单调递减函数，表述为：。

第和第的两只萤火虫位于和，它们之间的距离为笛卡尔距离（Cartesian distance），公式为：，式中为第个萤火虫空间坐标的第个分量。在2维情况下，公式为：。第只萤火虫被其它更有吸引力（更亮）的第只萤火虫吸引发生运动的公式为：，式中的第2项就是吸引力影响部分。第3项，是一个随机化，α

rand-1/2randβ0​=1，α∈[0,1]。

关键变量变化公式——光强度和吸引力的代码部分对应：

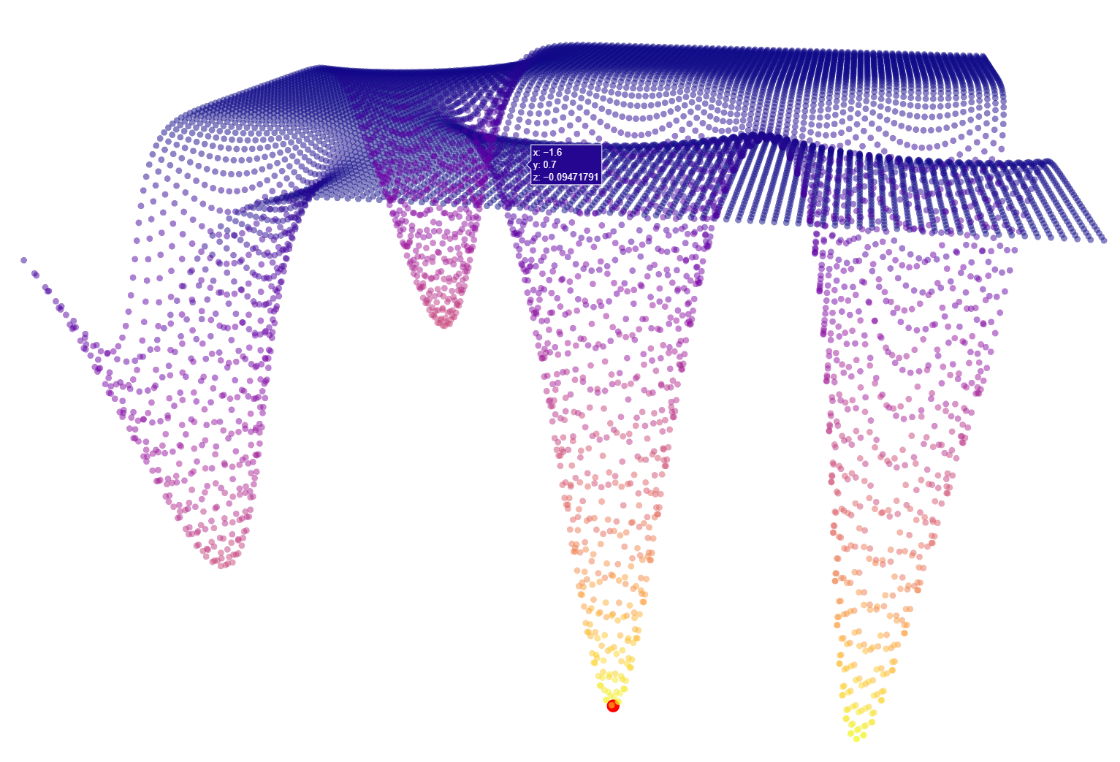
# Function: Distance Calculations  
def euclidean\_distance(x, y):  
 distance = 0  
 for j in range(0, len(x)):  
 distance = (x[j] - y[j])\*\*2 + distance   
 return distance\*\*(1/2)   
  
# Function: Beta Value  
def beta\_value(x, y, gama = 1, beta\_0 = 1):  
 rij = euclidean\_distance(x, y)  
 beta = beta\_0\*math.exp(-gama\*(rij)\*\*2)  
 return beta  
  
# Function: Ligth Intensity  
def ligth\_value(light\_0, x, y, gama = 1):  
 rij = euclidean\_distance(x, y)  
 light = light\_0\*math.exp(-gama\*(rij)\*\*2)  
 return light  
  
# Function: Update Position  
def update\_position(position, x, y, alpha\_0 = 0.2, beta\_0 = 1, gama = 1, firefly = 0, min\_values = [-5,-5], max\_values = [5,5], target\_function = target\_function):  
 for j in range(0, len(x)):  
 epson = int.from\_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1) - (1/2)  
 position[firefly, j] = np.clip((x[j] + beta\_value(x, y, gama = gama, beta\_0 = beta\_0)\*(y[j] - x[j]) + alpha\_0\*epson), min\_values[j], max\_values[j])  
 position[firefly, -1] = target\_function(position[firefly, 0:position.shape[1]-1])  
 return position

使用一个简单的4峰函数为基准函数检验 FA，函数公式为：，有，该函数有4个峰，2个局部峰位于和时，；2个全局峰位于和，有。因为定义的firefly\_algorithm函数为最小优化，因此对4峰函数值取反，从计算结果看到求得到了一个全局峰值，位置为，值为-2，与上述结果同。

def four\_peaks(variables\_values=[0, 0]):  
 x, y=variables\_values  
 func\_value=pow(math.e,-pow(x-4,2)-pow(y-4,2))+pow(math.e,-pow(x+4,2)-pow(y-4,2))+2\*(pow(math.e,-x\*\*2-y\*\*2)+pow(math.e,-x\*\*2-(y+4)\*\*2))  
 return -func\_value  
  
# FA - Parameters  
parameters = {  
 'swarm\_size': 50,  
 'min\_values': (-5, -5),  
 'max\_values': (5, 5),  
 'generations': 100,  
 'alpha\_0': 0.2,  
 'beta\_0': 1,  
 'gama': 1,  
 'verbose': 20  
}   
  
# FA - Algorithm  
fa,epoch=usda\_heuristic.firefly\_algorithm(target\_function=four\_peaks, \*\*parameters)   
  
# FA - Solution  
variables = fa[:-1]  
minimum = fa[ -1]  
print('Variables: ', np.around(variables, 4) , ' Minimum Value Found: ', round(minimum, 4) )

Generation: 0 f(x) = -1.554558799766898  
Generation: 20 f(x) = -1.9999920229090813  
Generation: 40 f(x) = -1.9999993359687203  
Generation: 60 f(x) = -1.9999993359687203  
Generation: 80 f(x) = -2.000000033061012  
Generation: 100 f(x) = -2.000000033061012  
Variables: [ 0.0003 -0.0002] Minimum Value Found: -2.0

# CS - Plot Solution  
plot\_parameters = {  
 'min\_values': (-5, -5),  
 'max\_values': (5, 5),  
 'step': (0.1, 0.1),  
 'solution': [variables],  
 'proj\_view': '3D',  
 'view': 'browser'   
}  
usda\_vis.plot\_single\_function(target\_function=four\_peaks, \*\*plot\_parameters)



注释（Notes）：

① pyMetaheuristic，元启发式算法库，（<https://github.com/Valdecy/pyMetaheuristic>）。

② mistre，（ <https://github.com/knaidoo29/mistree>）。

参考文献（References）:

[1] Segaran, T. Programming Collective Intelligence. (O’Reilly, 2007).

[2] Weiss, J. Landscape Ecology Principles In Landscape Architecture And Land Use Planning. in (2016).

[3] Tong, Z. A genetic algorithm approach to optimizing the distribution of buildings in urban green space. Autom Constr 72, 46–51 (2016).

[4] Clark, P. J. & Evans, F. C. Distance to Nearest Neighbor as a Measure of Spatial Relationships in Populations. Ecology 35, 445–453 (1954).

[5] How Average Nearest Neighbor works, <https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/h-how-average-nearest-neighbor-distance-spatial-st.htm>.

[6] What is a z-score? What is a p-value?, <https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/what-is-a-z-score-what-is-a-p-value.htm>

[7] Aerts, J. C. J. H. & Heuvelink, G. B. M. Using simulated annealing for resource allocation. International Journal of Geographical Information Science 16, 571–587 (2002).

[8] Carver, S., Evans, A., Kingston, R. & Turton, I. Public participation, GIS, and cyberdemocracy: Evaluating on-line spatial decision support systems. Environ Plann B Plann Des 28, 907–921 (2001).

[9] Pettit, C. J. Use of a collaborative GIS-based planning-support system to assist in formulating a sustainable-development scenario for Hervey Bay, Australia. Environ Plann B Plann Des 32, 523–545 (2005).

[10] Stewart, T. J., Janssen, R. & Van Herwijnen, M. A genetic algorithm approach to multiobjective land use planning. Comput Oper Res 31, 2293–2313 (2004).

[11] Janssen, R., van Herwijnen, M., Stewart, T. J. & Aerts, J. C. J. H. Multiobjective decision support for land-use planning. Environ Plann B Plann Des 35, 740–756 (2008).

[12] Cao, K., Huang, B., Wang, S. & Lin, H. Sustainable land use optimization using Boundary-based Fast Genetic Algorithm. Comput Environ Urban Syst 36, 257–269 (2012).

[13] Tsai, M. W., Hong, T. P. & Lin, W. T. A two-dimensional genetic algorithm and its application to aircraft scheduling problem. Math Probl Eng 2015, (2015).

[14] Prudius, Andrei A. Adaptive random search methods for simulation optimization. Georgia Institute of Technology ProQuest Dissertations Publishing,  2007. 3271580.

[15] Mirjalili, S. The ant lion optimizer. Advances in Engineering Software 83, 80–98 (2015).

[16] Abualigah, L., Diabat, A., Mirjalili, S., Abd Elaziz, M. & Gandomi, A. H. The Arithmetic Optimization Algorithm. Comput Methods Appl Mech Eng 376, (2021).

[17] Karaboga, D. An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report - TR06. Technical Report, Erciyes University (2005).

[18] 李晓磊,邵之江,钱积新.一种基于动物自治体的寻优模式:鱼群算法[J].系统工程理论与实践,2002(11):32-38.

[19] Yang, X.-S. A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Coop-erative Strategies for Optimization. vol. 284 (2010).

[20] Simon, D. Biogeography-based optimization. IEEE Transactions on Evolutionary Computation 12, 702–713 (2008).

[21] Rubinstein, R. Y. EUROPEAN JOURNAL OF OPERATIONAL Optimization of computer simulation models with rare events \*. European Journal of Operational Research vol. 99 (1997).

[22] Askarzadeh, A. A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. Comput Struct 169, 1–12 (2016).

[23] Yang, X.-S. & Deb, S. Cuckoo Search via Levy Flights. (2010) doi:10.48550/ARXIV.1003.1594.

[24] Storn, R. & Price, K. Differential Evolution-A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization vol. 11 (1997).

[25] al-Rifaie, M. M. & Aber, A. Dispersive flies optimisation and medical imaging. Studies in Computational Intelligence 610, 183–203 (2016).

[26] Meraihi, Y., Ramdane-Cherif, A., Acheli, D. & Mahseur, M. Dragonfly algorithm: a comprehensive review and applications. Neural Computing and Applications vol. 32 16625–16646 Preprint at https://doi.org/10.1007/s00521-020-04866-y (2020).

[27] Yang, X.-She. Nature-inspired optimization algorithms. Elsevier Inc, (2014).

[28] Karami, H., Valikhan Anaraki, M., Farzin, S. & Mirjalili, S. Flow Direction Algorithm (FDA): a Novel Optimizer Approach for Solving Optimization Problems. Comput Ind Eng 156, 107224 (2021).

[29] John H. Holland. Adaptation in Natural and Artificial Systems\_ An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence-MIT (1992) .

[30] Mirjalili, S., Mirjalili, S. M. & Lewis, A. Grey Wolf Optimizer. Advances in Engineering Software 69, 46–61 (2014).

[31] Saremi, S., Mirjalili, S. & Lewis, A. Grasshopper Optimisation Algorithm: Theory and application. Advances in Engineering Software 105, 30–47 (2017).

[32] Rashedi, E., Nezamabadi-pour, H. & Saryazdi, S. GSA: A Gravitational Search Algorithm. Inf Sci (N Y) 179, 2232–2248 (2009).

[33] Heidari, A. A. et al. Harris hawks optimization: Algorithm and applications. Future Generation Computer Systems 97, 849–872 (2019).

[34] Nadimi-Shahraki, M. H., Taghian, S. & Mirjalili, S. An improved grey wolf optimizer for solving engineering problems. Expert Syst Appl 166, (2021).

[35] Mostafa Bozorgi, S. & Yazdani, S. IWOA: An improved whale optimization algorithm for optimization problems. J Comput Des Eng 6, 243–259 (2019).

[36] Venkata Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. International Journal of Industrial Engineering Computations 7, 19–34 (2016).

[37] Chou, J. S. & Truong, D. N. A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean. Appl Math Comput 389, (2021).

[38] Bolaji, A. L. aro, Al-Betar, M. A., Awadallah, M. A., Khader, A. T. & Abualigah, L. M. A comprehensive review: Krill Herd algorithm (KH) and its applications. Applied Soft Computing Journal vol. 49 437–446 Preprint at https://doi.org/10.1016/j.asoc.2016.08.041 (2016).

[39] Moscato, P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. (1989).

[40] Mirjalili, S. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. Knowl Based Syst 89, 228–249 (2015).

[41] Mirjalili, S., Mirjalili, S. M. & Hatamlou, A. Multi-Verse Optimizer: a nature-inspired algorithm for global optimization. Neural Comput Appl 27, 495–513 (2016).

[42] Yapici, H. & Cetinkaya, N. A new meta-heuristic optimizer: Pathfinder algorithm. Applied Soft Computing Journal 78, 545–568 (2019).

[43] Kennedy, J. & Eberhart, R. Particle swarm optimization. in Proceedings of ICNN’95 - International Conference on Neural Networks vol. 4 1942–1948 vol.4 (1995).

[44] Anderson, R. L. RECENT ADVANCES IN FINDING BEST OPERATING CONDITIONS\*. Source: Journal of the American Statistical Association vol. 48 (1953).

[45] Mirjalili, S. et al. Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. Advances in Engineering Software 114, 163–191 (2017).

[46] Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. Optimization by Simulated Annealing Downloaded from. vol. 220 http://science.sciencemag.org/ (1983).

[47] Mirjalili, S. SCA: A Sine Cosine Algorithm for solving optimization problems. Knowl Based Syst 96, 120–133 (2016).

[48] Das, B., Mukherjee, V. & Das, D. Student psychology based optimization algorithm: A new population based optimization algorithm for solving optimization problems. Advances in Engineering Software 146, (2020).

[49] Cheng, M. Y. & Prayogo, D. Symbiotic Organisms Search: A new metaheuristic optimization algorithm. Comput Struct 139, 98–112 (2014).

[50] Rao, R. V., Savsani, V. J. & Vakharia, D. P. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. CAD Computer Aided Design 43, 303–315 (2011).

[51] Mirjalili, S. & Lewis, A. The Whale Optimization Algorithm. Advances in Engineering Software 95, 51–67 (2016).