Simple-RS-CMSA-ES：带有禁忌区域的自适应协方差演化策略

孙智聪
计算机科学与工程系
南方科技大学
深圳，中国
12032471

摘要—多波峰优化是演化计算的一大重要研究领域，目的是找到搜索空间中的所有全局或者局部最优解。本工作的目的是参加 GECCO 2021 竞赛——用 Niching 方法解决多波峰优化问题。文中提出了一种简单的带有禁忌区域的自适应协方差演化策略算法 (Simple-RS-CMSA-ES)，算法的核心思想是利用 CMSA-ES 去更新每个 Niche，并使用禁忌区域来保证 Exploitation 和 Exploration。在此基础上，本工作将 GECCO 2021 的 20 个函数按特点分为三组，进行三个实验，每个函数都进行了 50 次独立重复的运行，并使用平均召回率也即 PR、静态 F1 measure 和动态 F1 measure 作为评价指标。结果表明：对于低维度或者较少波峰的函数，算法的基本能搜索到所有波峰，而随着波峰数目或者维度的增加，由于局部搜索能力的限制，算法的 PR 值有所下降。特别的，难度最大的两个函数为 function 9 和 function 20，218 个波峰的 function 9 上 PR 值为 0.4，在 20 维的 function 20 上 PR 为 0.35。但是此算法的整体表现与本课程大部分其他同学相比还是较优的。

***Index Terms*—多波峰优化，Niching，CMSA-ES, Taboo area, Exploitation, Exploration**

I. 介绍

本文将在 Section I 中简要介绍多波峰优化和 Niching 的概念，并详细介绍使用到的 20 个 benchmark 函数的特点。Section II 中将详细介绍本工作的算法，即简单的带有禁忌区域的自适应协方差算法，并分析其中的原理与使用动机。Section III 是实验部分，将待测函数分为低维较少波峰、低位较多波峰和高维较多波峰三类，进行三组实验，并讨论分析了实验结果。最后 Section IV 中将总结本文的工作，并提出解决目前算法可能存在的问题的可能方案和未来工作。

Mutil-modal opimization 即多模态，目标是找到搜索空间中所有的全局或者局部的最优解，而 Niching 小生境是解决这一问题最主要的方法，它的核心思想是将整个大种群分为一个个 Niche，每个 Niche 中有一个子种群，每个 Niche 各自进行演化且搜索区域尽量不互相重叠，因此既保证算法能够在最优点附近进行 Exploitation，不断搜小范围收敛到真正的最优，又能保持多样性，在为探索的区域进行 Exploration，尽量找到更多的最优点。

本文以参加 GECCO 2021 Competition [1] 为目的，GECC0 2021 提供了 20 个 benchmark 函数，这 20 个函数的名字及其维度如下所示，括号为函数的维度情况，例如 F6 为一种函数类型在 2D 和 3D 下对应的两个函数：

- F1: Five-Uneven-Peak Trap (1D)
- F2: Equal Maxima (1D)
- F3: Uneven Decreasing Maxima (1D)
- F4: Himmelblau (2D)
- F5: Six-Hump Camel Back (2D)
- F6: Shubert (2D, 3D)
- F7: Vincent (2D, 3D)
- F8: Modified Rastrigin - All Global Optima (2D)
- F9: Composition Function 1 (2D)
- F10: Composition Function 2 (2D)
- F11: Composition Function 3 (2D, 3D, 5D, 10D)
- F12: Composition Function 4 (3D, 5D, 10D, 20D)

这些函数在一位或者二维的图像如图 1 至图 12 所示，图源自 [1]。函数有如下特点：

- F1 到 F3 是简单的一维度函数；
- F4 和 F5 是简单二维函数，但是 not scalable;
- F6 到 F8 是 scalable 的多峰函数，其中 F6 和 F7 的山峰数目随着维度增加而增加；
- F9 到 F12 是由多个基础函数组合而成的，其中 F9 和 F10 是 separable 并且 non-symmetric，而 F11 和 F12 则是 non-separable 且 non-symmetric，这几个函数的波峰个数与维度无关；

除此之外，函数的全局最优点个数也是一大关键，每个函数的全局最优点个数统计如图 13 所示，图源自 [1]。

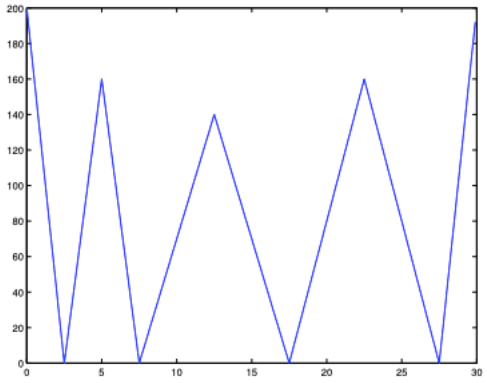

图 1. Function 1

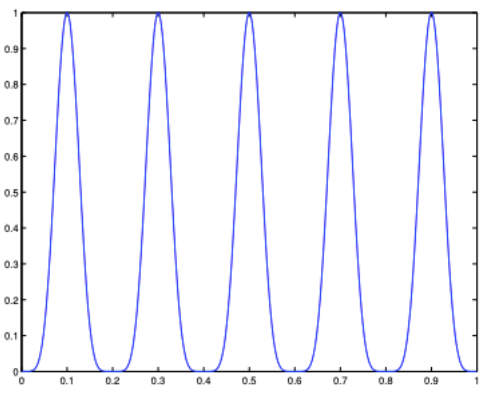

图 2. Function 2

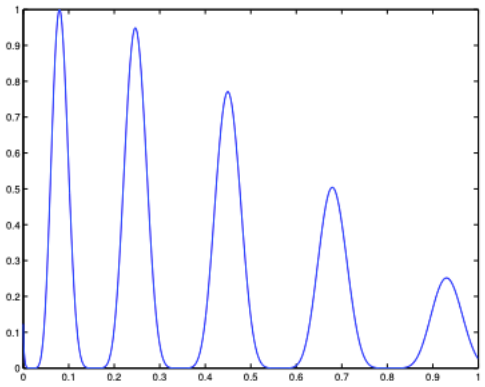

图 3. Function 3

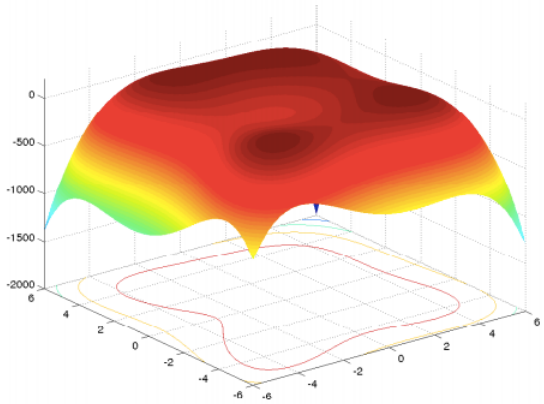

图 4. Function 4

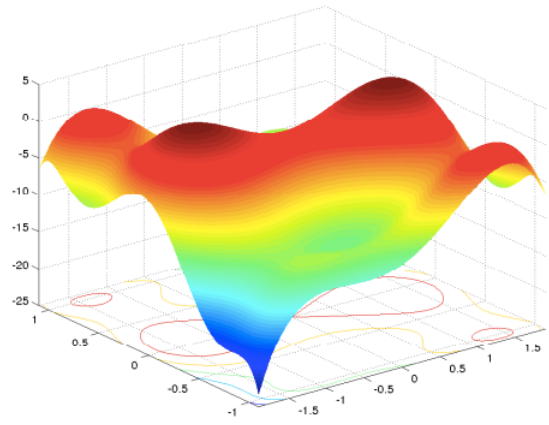

图 5. Function 5

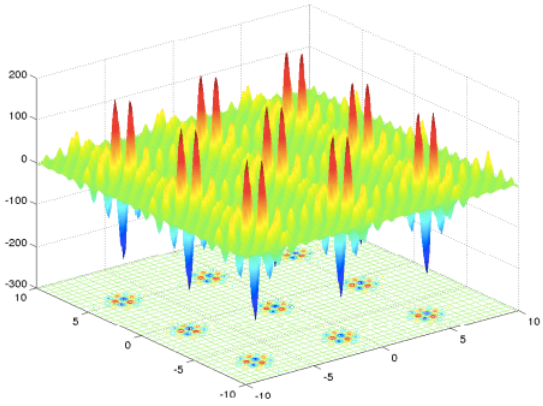

图 6. Function 6

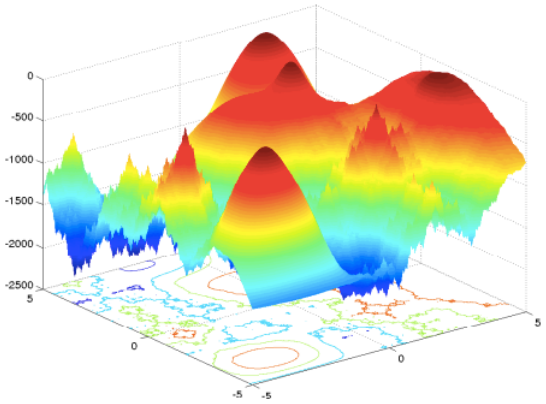

图 9. Function 9

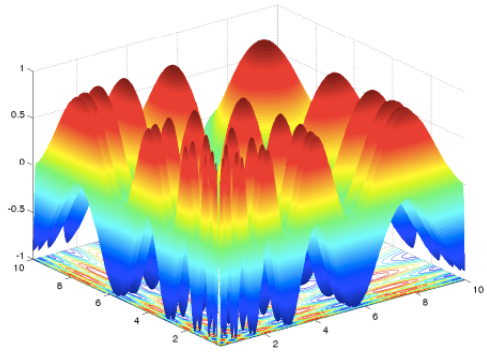

图 7. Function 7

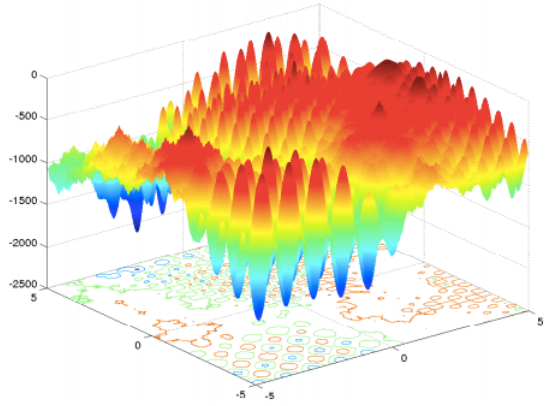

图 10. Function 10

V

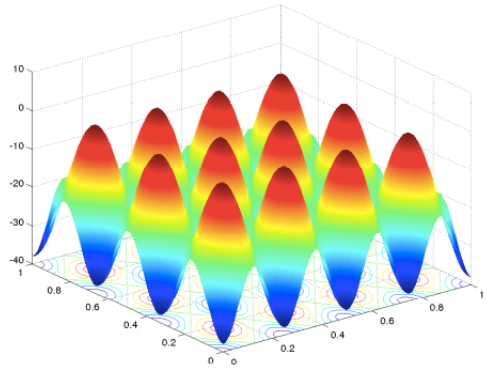

图 8. Function 8

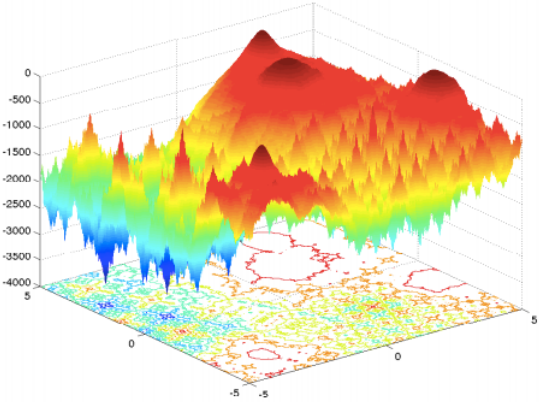

图 11. Function 11

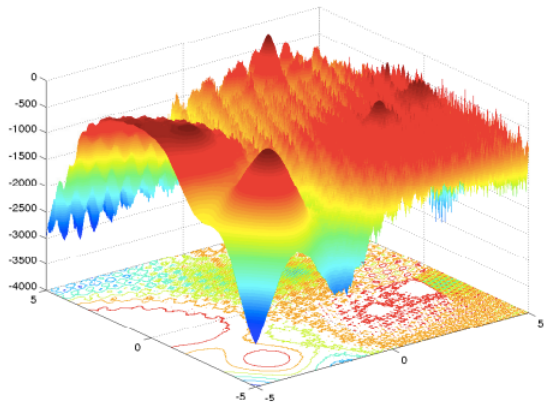

图 12. Function 12

Function	r	Peak height	No. global optima
F_1 (1D)	0.01	200.0	2
F_2 (1D)	0.01	1.0	5
F_3 (1D)	0.01	1.0	1
F_4 (2D)	0.01	200.0	4
F_5 (2D)	0.5	1.031628453	2
F_6 (2D)	0.5	186.7309088	18
F_7 (2D)	0.2	1.0	36
F_6 (3D)	0.5	2709.093505	81
F_7 (3D)	0.2	1.0	216
F_8 (2D)	0.01	-2.0	12
F_9 (2D)	0.01	0	6
F_{10} (2D)	0.01	0	8
F_{11} (2D)	0.01	0	6
F_{11} (3D)	0.01	0	6
F_{12} (3D)	0.01	0	8
F_{11} (5D)	0.01	0	6
F_{12} (5D)	0.01	0	8
F_{11} (10D)	0.01	0	6
F_{12} (10D)	0.01	0	8
F_{12} (20D)	0.01	0	8

图 13. Function 12

II. 带有拒绝域的自适应协方差演化策略

A. 算法概述

本文使用到算法 Simple-RS-CMSA-ES，全称为简单的排斥子种群的自适应协方差演化策略 (Simple-Self-Adaptation Evolution Strategy with Repelling Subpopulations)，也可以称为带禁忌区域 (Taboo Area) 的自适应协方差演化策略 (TA-CMSA-ES)，本文中统一用 Simple-RS-CMSA-EA 表示。算法在每个测试函数上完整运行一次的过程如伪代码 Simple-RS-CMSA-ES 1:One-Run 所示。其**主要步骤**如下：

1) 首先在函数取值范围内随机均衡地生成一个初始种群，其中的每个初始个体都是一个独立 Niche 的初始中心。下一代 Niche 将围绕着这些中心（初始个体）产生。每个 Niche 都围绕着它的初始中心，初始化一个拒绝域半径，以该半径画圆得到的就是该 Niche 的禁忌区域。关于禁忌点和禁忌区域的更多细节将在中介绍。

Simple-RS-CMSA-ES 1: One-Run

Input: Function information

Output: Optimal solution set Archive

Obtain the dimension D and of current function;
Set population size *popsize* to 50 times of D;
Set the rest number of evaluation *evaluationremainedcnt* to maximum number of evaluation;
Initialize taboo points and reject region but not assign values;
while *evaluation remained cnt* > 0 **do**
    Initialization(or Restar) of population, each individual is a niche which is not in the reject region of any taboo points;
    Evaluate the initial individuals and sort them in descending order of fitness;
    evaluation remained cnt = evaluation remained cnt - pop size;
    for *each individual in population* **do**
        Perform RS-CMSA-ES on this individual to get new population;
        Use best individual in new population to update taboo point and reject region;
        Use new population to update Archive
    end
    return *Archive*
end

2) 评估初始化种群，并将种群中的个体按适应度降序排列。目的是让适应度更高的 Niche 先被更新。

2) 遍历每个 Niche，在每个 Niche 上，执行以下操作：

- 围绕着初始中心（初始个体）使用 RS-CMSA-EA 进行演化产生一群足够好的新个体，且要求新的个体不在其他 Niche 禁忌区域。新个体和初始个体构成该 Niche 的总的子种群。

- 使用该子种群中适应度最优的个体更新该 Niche 的禁忌点（中心）和禁忌区域。
- 使用子种群更新 Archive。

3) 若适应度评价次数使用完，则输出 Archive。否则进行种群的再次初始化，并执行以上的 2) 和 3) 步骤。此时新生成的 Niche 初始中心不能在上一代所有 Niche 的禁忌区域中，目的是减少此次搜索区域和之前搜索区域的重叠，扩大搜索空间，以便搜索更多可能的全局最优点。

通过以上思路可以看出，本算法的**核心**有 4 部分：

1) 随机均匀地生成初始 Niche 中心，使得算法能够更广地覆盖函数取值范围。

2) 使用 RS-CMSA-ES 去更新每个 Niche 直到该 Niche 不能再被优化。

3) 禁忌区域的使用：避免 Niche 搜索区域的重叠 (Exploitation)，使用重新初始化的种群探索未开发区域 (Exploration)。

4)Archive 的更新策略。

接下来就针对这 4 点核心的 2、3、4 进行详细的介绍。

B. RS-CMSA-ES

自适应协方差演化策略是在简单的高斯演化策略基础上加入协方差的自适应更新得到的，而 RS-CMSA-ES 则是 CMSA-ES 的基础上进一步加入了禁忌区域的限制。本部分将一步步详细介绍该算法。

假设我们想优化一个函数 f(x) 但 f(x) 的梯度难以求解 [5]。只能得到在特定 x 下 f(x) 的确定值。我们认为随机变量 x 的概率分布 $p_\theta(x)$ 是函数 f(x) 优化问题的一个较优的解，θ 是分布的参数。在给定固定分布形式（例如，高斯分布）的情况下，参数 θ 包含了最优解的知识，在一代与一代间进行迭代更新。我们最终的目标是找到 θ 的最优设置。而简单高斯进化策略是进化策略的最基础和最经典的版本 [5]。它将 $p_\theta(x)$ 建模为一个 n 维的各向同性的高斯分布，在高斯分布中 θ 指的是均值 μ 和标准差 σ。

$$\theta = (\mu, \sigma), p_\theta(x) \sim \mathcal{N}\left(\mu, \sigma^2 I \sim \mu + \sigma\mathcal{N}(0, I)\right) \quad (1)$$

简单高斯进化策略的步骤如下，给定 $x \in R^n$

1）给定初始化参数 $\theta = \theta_0$，种群代数 $t = 0$。

2）从高斯分布中采样大小为 λ 的后代种群：

$$D^{(t+1)} = \left\{x_i^{(t+1)} \mid x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)}\right\} \quad (2)$$

其中，

$$y_i^{(t+1)} \sim \mathcal{N}(x \mid 0, \mathbf{I}), i = 1, \ldots, \lambda \quad (3)$$

3）在后代种群中使用精英策略，选择适应度排名靠前的 μ 个个体作为精英集，并标注为：

$$D_{\text{elite}}^{(t+1)} = \left\{x_i^{(t+1)} \mid x_i^{(t+1)} \in D^{(t+1)}, i = 1, \ldots, \mu, \mu \leq \lambda\right\} \quad (4)$$

4）使用精英集为下一代种群更新新的均值和方差：

$$\mu^{(t+1)} = \text{avg}\left(D_{\text{elite}}^{(t+1)}\right) = \frac{1}{\mu}\sum_{i=1}^{\mu} x_i^{(t+1)} \quad (5)$$

$$\sigma^{(t+1)^2} = \text{var}\left(D_{\text{elite}}^{(t+1)}\right) = \frac{1}{\mu}\sum_{i=1}^{\mu}\left(x_i^{(t+1)} - \mu^{(t)}\right)^2 \quad (6)$$

5）重复步骤 2）到 4）直至种群达到要求的相对最优。

在简单高斯演化策略中，$\sigma(t+1)$ 与 $\sigma(t)$ 密切相关，因此算法不能在需要时（即置信度改变时）迅速调整探索空间。而 CMSA-ES 通过使用协方差矩阵 C 跟踪分布上得到的样本两两之间的依赖关系，解决了这一问题。新的分布参数变为了：

$$\theta = (\mu, \sigma, C), p_\theta(x) \sim \mathcal{N}\left(\mu, \sigma^2 C\right) \sim \mu + \sigma\mathcal{N}(0, C) \quad (7)$$

其中，σ 控制分布的整体尺度，我们通常称之为步长。C 即协方差矩阵。其更新方式如公式 (8) -(11) 所示 [2]。其中 λ 为种群大小，在本工作中即为每个 Niche 的种群大小，N_{elt} 是父代种群的精英个数，τ_c 为 C 的学习率，μ 是用来更新种群均值和方差的个体数目。

$$\mathbf{C}_i \leftarrow \left(1 - \frac{1}{\tau_c}\right)\mathbf{C}_i + \frac{1}{\tau_c}\sum_{j=1}^{\mu} w_j \left(z_{ij}^T z_{ij}\right) \quad (8)$$

$$\sigma_{\text{mean}\ i} \leftarrow \frac{\sigma_{\text{mean}\ i}\left(\prod_{j=1}^{\mu}\sigma_{ij}^{w_j}\right)}{\left(\prod_{j=1}^{\lambda+N_{\text{elt}}}\sigma_{ij}^{\frac{1}{\lambda+N_{\text{elt}}}}\right)} \quad (9)$$

$$\boldsymbol{x}_{\text{mean}\ i} \leftarrow \sum_{i=1}^{\mu} w_j \boldsymbol{x}_{ij} \quad (10)$$

$$\omega_j = \frac{\ln(\mu+1) - \ln(j)}{\sum_{j=1}^{\mu}(\ln(\mu+1) - \ln(j))}, j = 1, 2, \ldots, \mu \quad (11)$$

Simple-RS-CMSA-ES 2: RS-CMSA-ES

Input: A niche center x_{mean}, taboo point set, rest number of evaluation

Output: New population, evaluation remained cnt

Difne initial subpopulation size = $[4.5 + 3 * log(D)]$, (D is the dimension of this function);
Difine size of individuals used for update covariance μ;
Difine number of elites = $max(1, [0.15 * \mu])$;Difine Max search count = $10 + [30 * D/\mu]$;
Difne local search count = 0;Initialize best score as the fitness of this niche center;
while *local search count < Max search count and evaluation remained cnt > 0* **do**
    if *This niche center is in the reject region of the nearest taboo point* **then**
        return *This niche center, evaluation remained cnt;*
    end
    Generate new population using this niche center;
    Evaluate new population;
    Combine new population and this niche center as total population;
    Sort total population in descending order of fitness, and update best score;
    if *best score of total population > best score* **then**
        Update best score,local search count = 0;
    end
    else
        local search count ++;
    end
    Update w_j, x_{mean}, C_i, σ_{mean};
    if *$C_i$has a negative eigenvalue* **then**
        return *This niche center, evaluation remained cnt;*
    end
end
return *This niche center, evaluation remained cnt;*

C. 禁忌区域及其使用

本文每个 Niche 中都使用最优个体作为当前 Niche 的禁忌点，即禁忌点个数等于 Niche 个数。

禁忌区域的初始化：初始 Niche 的禁忌点即为初始种群中的个体，不设置禁忌半径。

禁忌区域的更新：在每个 Niche 更新过程中，禁忌点在遍历每个 Niche 时 (步骤 2))，将每个 Niche 通过 CMSA-ES 更新得到当前 Niche 的最优的子种群，并使用最优子种群中的最优个体替代旧的禁忌点。而新禁忌点和旧的禁忌点之间的距离即为新的禁忌区域的半径。

禁忌区域主要使用在两个地方：

1) 在每个 Niche 中更新子种群时，需要通过对多元高斯分布不断采样获取新的子个体，而子个体有可能在其他 Niche 的禁忌区域内，这会造成 Niche 的搜索空间的重叠，不利于 Exploitation。因此本算法在生成新个体后会进行判断，若新的子个体在其他 Niche 的禁忌内，则抛弃该个体，重新生成新的个体，直至个体数达到要求。

2) 在对所有 Niche 更新完毕后，每个 Niche 的搜索结果已经达到了相对最优（经多多次演化，Niche 最优个体适应度没有变化）。适应度评估次数若还有剩余，则进行新一轮的 Niche 初始化。此时如果向第一次初始化那样均匀随机初始化 Niche 中心（初始个体），则新的一次全局搜索结果可能与第一次全局搜索结果类似，因此，需要使用之前最后一次更新得到的 Niche 的禁忌区域，在非禁忌区域上生成新一代 Niche 初始中心。在此设置下，新一代的种群能够在搜索初期尽量避开之前搜索过的区域（尽管有部分区域可能会重叠，但是少部分），因此扩大了搜索空间，增大搜索到其他最优点的可能性。

D. Archive 更新策略

Archive 更新是在遍历每个 Niche 的时候进行的，用来保存找到的所有最优策略。其**主要步骤**如下：

1) 首先设置新的 Archive 为空。

2) 如果是对初始化种群（或再次初始化对种群）中的 Niche 更新 Archive，先使用 Hill-valley 函数判断当前 Niche 中的个体是否和旧的 Archive （这里的旧 Archive 指的是上一次更新后的 Archive）中的解在同一个盆地 basin，否则加入新的 Archive 中。

3) 如果是对非初始化种群的 Niche 更新 Archive，则：

- 首先更新全局最优个体的标准：记录当前 Niche 和旧的 Archive 中适应度最大值。
- 用旧的 Archive 更新新的 Archive：将旧的 Archive 中适应度与当前最大值在一定范围（如 0.1）内的个体加入新的 Archive。
- 用当前 Niche 更新 Arichive：遍历当前 Niche 中所有个体，将适应度与当前最大值在一定范围（如 0.1）内的个体加入新的 Archive。

Archive 更新算法中用来测试两个个体是否在同一盆地的函数为 Hill-valley，如图 14 所示。Hill-valley 函数利用 basin 的特点为的判断依据：

1) 当两个点用于判断的点在同一 basin 中，如 e_1 和 e_2，则在两点之间对适应度函数进行等间隔采样，此时这些中间的 i_1、i_2 和 i_3 的适应度函数值应该比 e_1 和 e_2 大。

2) 当两个点用于判断的点在不同 basin 中，如 e_3 和 e_4，则在两点之间对适应度函数进行等间隔采样，此时这些中间的 i_4、i_5 和 i_6 应该存在点的适应度值是小于 e_3 和 e_4 的，比如图中的 i_5。

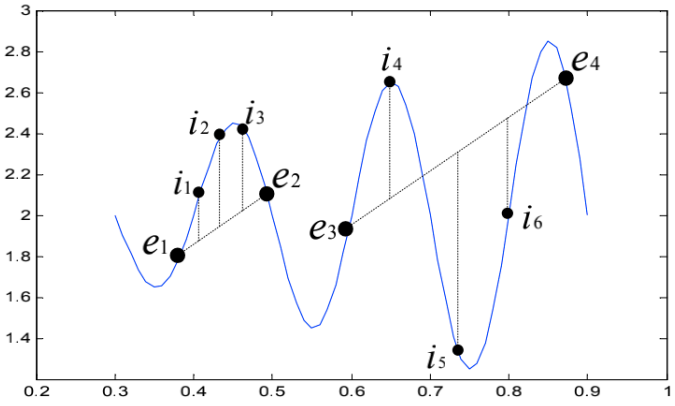

图 14. 一维 Hill-velley 函数示意图

具体算法的伪代码见 Simple-RS-CMSA-ES 3: Update-Archive Part I 和 Part II：

Simple-RS-CMSA-ES 3: Upadate-Archive Part I

Input: Old archive, New population, Rest number of evaluation

Output: New archive, Rest number of evaluation

```
Sort the new population in descending order and update it;
if Is the initial population then
    for each individual in population do
        Use Hill-valley function to junge if this individual in new population and any individual in old archive are in the same basin;
        if this individual in new population and any individual in old archive are not in the same basin then
            Add the solution of this individual into initial archive;
            Update the rest number of evaluation;
        end
    end
end
```

III. 实验与讨论

A. 实验设置

本实验的算法在 GECCO 2021 提供的 20 个多波峰函数上进行 50 次独立重复的运行，每个函数都有固定的 fitness 评估次数上限。

GECCO 用于评价算法在**每个函数上**的效果的指标有三个 [4]，这三个指标分别是：

- 召回率 Recall：即找到的不重复波峰数目占该函数所有波峰数的比例
- 静态 F1 分数 Static F1 measure：即 Recall 和 Precision 的乘积，其中 Precision 指的是算法找到真正的不重复波峰数占所有解的比例
- 动态 F1 分数 Dynamic F1 measure：即以静态 F1 为纵轴，个体评估次数为横轴的曲线上，静态 F1 的 AUC 值除以总的评估次数。

这些指标都是对 **50 次 Run 的平均值**，其中第一个指标，即 Recall 的平均值又可以称为 PR（Peak ratio）值。而最终对比不同算法的时候，GECCO 将计算每个指标在所有 benchmark 上的平均值，首先使用

Simple-RS-CMSA-ES 4: Upadate-Archive Part II

```
else
    Get the individual which has the best fitness from the old archive and new population, record its fitness value as Max score;
    for solution in old archive do
        if The difference between fitness of solution and Max score is in a certain range then
            Add solution into new archive;
        end
    end
    for individual in new population do
        if The difference between fitness of individual and all solutions in new archive is in a certain range then
            Use Hill-valley function to junge if this individual in new population and any individual in new archive are in the same basin;
            if this individual in new population and any individual in new archive are not in the same basin then
                Add individual into new archive;
            end
        end
    end
    Update the rest number of evaluation;
end
```

第一个指标进行排名，第一个指标排名相同的再使用第二个指标，以此类推。

在本实验中，由于该比赛之前的排名使用前两个指标就能分出完整排名，因此本实验主要考虑前两个指标，即**召回率和静态 F1 分数**。

根据之前对 benchmark 函数特点的讨论，本实验将分 3 组，分别研究算法在低维少波峰、低维多波峰、高维多波峰三类函数上的搜索效果，这三组实验包含的函数分别为：

- 实验一：包含函数 F1 到 F5，对应 function 1 至 function 5，主要是低维度，且全局最优点（波峰）较少的函数。
- 实验二：包含函数 F6 到 F10 的全部，以及 F11 和 F12 的低维 function，对应 function 6 至 function 15 主要是低维度多波峰函数。
- 实验三：包含函数 F11 和 F12 的高维 function，包括 5 维度、10 维度、20 维度，对应 function 16 至 function 20。这些函数的波峰较多且维度高。

除此之外，根据 GECCO 2021 的要求，本实验结果的精度分为 5 个级别，分别是 10^{-1}、10^{-2}、10^{-3}、10^{-4}、10^{-5}，精度是用来判断算法得出的解是否为全局最优解的阈值，精度越高（值越小）则解被判断为全局最优的可能性越低。

注：本工作的代码基于官方提供的 python 模块 [3]，算法加入禁忌区域的想法（非具体算法）以及 CMSA-EA 算法的参数设置思路借鉴自 [2]，代码为本人完成，非修改其他代码。

B. 实验结果与讨论

所有实验的统计结果如表格 I 和表格 II 所示，表格 I 是算法在每个 function 上进行 50 次 Run 的 Recall 平均值（即 PR 值）和方差，表格 II 是是算法在每个 function 上进行 50 次 Run 的 Static F2 measure 平均值和方差，两个表格都统计了 5 个不同精度下的情况。

为了方便分析算法在不同 function 下的运行结果，本工作绘制了算法在不同 Accuracy level 下 50 次 Recall 和 Static F1 measure 的箱图。其中，箱图的使用如图 15 所示，图源自 [4]。

1）实验一：实验一是在 function 1 至 function 5 上进行的，即低维且较少波峰函数，其 Recall 和 Static F1 measure 的箱图如图 16 至图 25 所示。算法在除了 function 2 的函数上能找到所有波峰，在 function 2 函数上即使精度要求到 10^{-5} 找到波峰数也达到 0.97 以上，这表明算法对于低维且较少波峰的函数具有很强的搜索能力。

2）实验二：实验二是在 function 6 至 function 15 上进行的，即低维且较多波峰函数，其 Recall 和 Static F1 measure 的箱图如图 26 至图 45 所示。其中 function 6 至 function 10 是波峰数最多的几个，function 9 更是

达到 218 个波峰。算法在 function 6 和 functioin 10 都找到了 0.97 以上的波峰数，而 function 7 和 function 8 分别为 0.71 和 0.77 左右。function 9 只能找到最多 0.44（精度 0.1 情况下）的波峰数，而且观察数据可以发现，随着精度的要求提高，在 function 9 上找到的比例不断下降了 10 个百分点，这表明算法确实搜索到了最优值的附近，但是可能由于搜索的步长较大，搜索粒度不够细，不能够更加靠近最优策略，这是局部搜索能力受限导致的。而算法在 function 11 至 function 15 上表现效果较好故不再赘述。

3) 实验三：实验三是在 function 16 至 function 20 上进行的，即高维且较多波峰函数，其 Recall 和 Static F1 measure 的箱图如图 46 至图 55 所示。算法在 function 16 和 functioin 18 找到的波峰数分别约为 0.69、0.74、0.67，而在 function 19 和 function 20 则降到了 0.49 和 0.35。这表明随着维度的提高，算法的搜索能力出现明显下降。而且观察数据可以发现，算法找到的波峰数目并没有随着 Accuracy level 的提高而改变（最多改变 0.01）这说明，随着要求精度的提高，找到的波峰数目没有多大变化，分析原因：算法在最优策略附近能够较好的收敛，但是由于本工作定义的禁忌区域（每个 Niche 的形状）为圆，由圆形相切的知识可以知道，不同 Niche 之间会存在搜索的盲区，即有一些带有最优策略的区域可能不在所有 Niche 的搜索空间内，因此没有被搜索到。

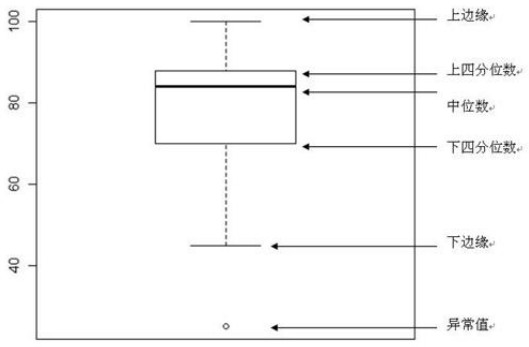

图 15. 箱图示意图

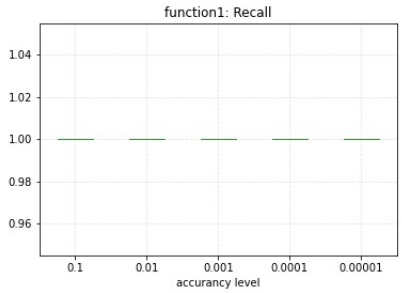

图 16. Function 1: Recall

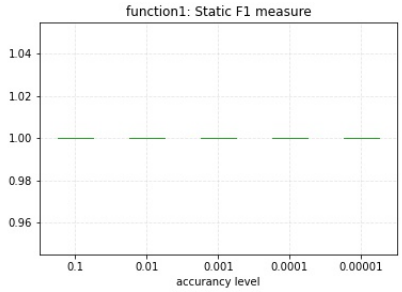

图 17. Function 1: Static F1 score

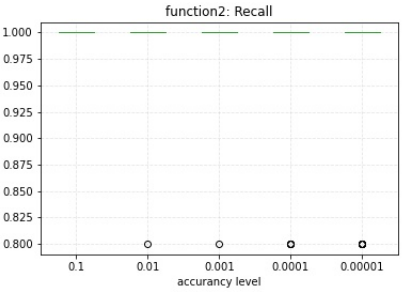

图 18. Function 2: Recall

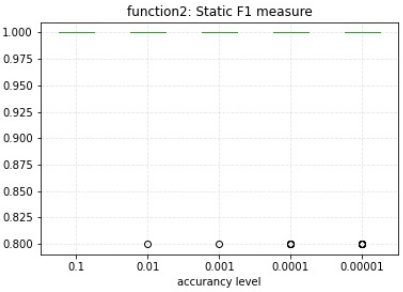

图 19. Function 2: Static F1 score

Accuracy lecel	function 1		function 2		function 3		function 4		function 5	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.01000	1.00000	0.00000	0.99600	0.00080	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.00100	1.00000	0.00000	0.99600	0.00080	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.00010	1.00000	0.00000	0.98400	0.00300	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.00001	1.00000	0.00000	0.97600	0.00431	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
Accuracy lecel	function 6		function 7		function 8		function 9		function 10	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	0.99889	0.00006	0.83944	0.00187	0.78025	0.00243	0.44694	0.00051	1.00000	0.00000
0.01000	0.99444	0.00028	0.76333	0.00259	0.77481	0.00270	0.40315	0.00040	0.98167	0.00150
0.00100	0.99000	0.00046	0.71389	0.00272	0.77235	0.00273	0.37991	0.00045	0.97500	0.00177
0.00010	0.98667	0.00057	0.69111	0.00289	0.76938	0.00260	0.35991	0.00051	0.97167	0.00187
0.00001	0.98556	0.00061	0.67833	0.00325	0.76469	0.00260	0.34602	0.00044	0.97000	0.00192
Accuracy lecel	function 11		function 12		function 13		function 14		function 15	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	1.00000	0.00000	0.89250	0.00383	0.83667	0.00736	0.83667	0.00056	0.75000	0.00000
0.01000	0.99333	0.00109	0.88750	0.00462	0.83333	0.00680	0.83667	0.00056	0.74750	0.00031
0.00100	0.99333	0.00109	0.88250	0.00473	0.83333	0.00680	0.83000	0.00169	0.74250	0.00090
0.00010	0.98667	0.00209	0.88000	0.00444	0.83333	0.00680	0.83000	0.00169	0.73750	0.00207
0.00001	0.98667	0.00209	0.88000	0.00444	0.83333	0.00680	0.83000	0.00169	0.73500	0.00232
Accuracy lecel	function 16		function 17		function 18		function 19		function 20	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	0.69667	0.00418	0.74750	0.00031	0.66667	0.00000	0.49750	0.00159	0.35000	0.00765
0.01000	0.69667	0.00418	0.74500	0.00061	0.66667	0.00000	0.49750	0.00159	0.35000	0.00765
0.00100	0.69000	0.00341	0.74000	0.00117	0.66667	0.00000	0.49750	0.00159	0.35000	0.00765
0.00010	0.68667	0.00413	0.74000	0.00117	0.66667	0.00000	0.49750	0.00159	0.35000	0.00765
0.00001	0.68333	0.00482	0.73750	0.00143	0.66667	0.00000	0.49000	0.00245	0.35000	0.00765

表 I

Recall

Accuracy lecel	function 1		function 2		function 3		function 4		function 5	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	1.00000	0.00000	1.00000	0.00000	0.50000	0.00000	1.00000	0.00000	1.00000	0.00000
0.01000	1.00000	0.00000	0.99600	0.00080	0.50000	0.00000	1.00000	0.00000	1.00000	0.00000
0.00100	1.00000	0.00000	0.99600	0.00080	0.50000	0.00000	1.00000	0.00000	1.00000	0.00000
0.00010	1.00000	0.00000	0.98400	0.00300	0.50000	0.00000	1.00000	0.00000	1.00000	0.00000
0.00001	1.00000	0.00000	0.97600	0.00431	0.50000	0.00000	1.00000	0.00000	1.00000	0.00000
Accuracy lecel	function 6		function 7		function 8		function 9		function 10	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.01000	0.99556	0.00023	0.90963	0.00215	0.99285	0.00011	0.90281	0.00143	0.98167	0.00150
0.00100	0.99111	0.00042	0.85126	0.00353	0.98967	0.00015	0.85084	0.00182	0.97500	0.00177
0.00010	0.98778	0.00054	0.82414	0.00380	0.98598	0.00020	0.80614	0.00232	0.97167	0.00187
0.00001	0.98667	0.00057	0.80875	0.00413	0.98000	0.00028	0.77523	0.00235	0.97000	0.00192
Accuracy lecel	function 11		function 12		function 13		function 14		function 15	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	0.77662	0.03890	0.38156	0.02230	1.00000	0.00000	0.98667	0.00209	1.00000	0.00000
0.01000	0.77106	0.03867	0.37954	0.02251	0.99667	0.00056	0.98667	0.00209	0.99667	0.00056
0.00100	0.77106	0.03867	0.37777	0.02272	0.99667	0.00056	0.97867	0.00344	0.99000	0.00160
0.00010	0.76551	0.03837	0.37705	0.02288	0.99667	0.00056	0.97867	0.00344	0.98333	0.00368
0.00001	0.76551	0.03837	0.37705	0.02288	0.99667	0.00056	0.97867	0.00344	0.98000	0.00413
Accuracy lecel	function 16		function 17		function 18		function 19		function 20	
	mean	var	mean	var	mean	var	mean	var	mean	var
0.10000	0.98600	0.00345	0.99500	0.00125	0.45860	0.01133	1.00000	0.00000	1.00000	0.00000
0.01000	0.98600	0.00345	0.99167	0.00177	0.45860	0.01133	1.00000	0.00000	1.00000	0.00000
0.00100	0.97800	0.00479	0.98500	0.00275	0.45860	0.01133	1.00000	0.00000	1.00000	0.00000
0.00010	0.97467	0.00746	0.98500	0.00275	0.45860	0.01133	1.00000	0.00000	1.00000	0.00000
0.00001	0.96967	0.00845	0.98250	0.00426	0.45860	0.01133	0.98500	0.00360	1.00000	0.00000

表 II

Static F1 score

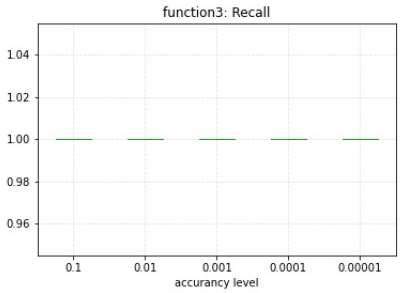

图 20. Function 3: Recall

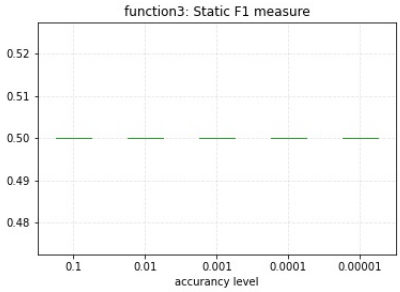

图 21. Function 3: Static F1 score

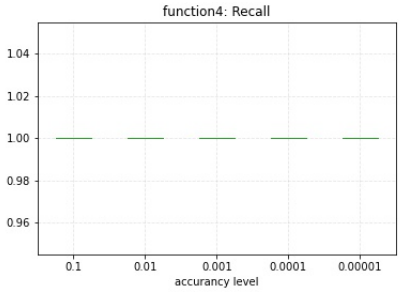

图 22. Function 4: Recall

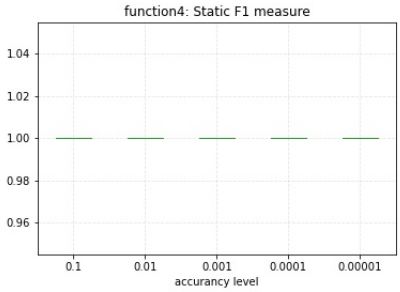

图 23. Function 4: Static F1 score

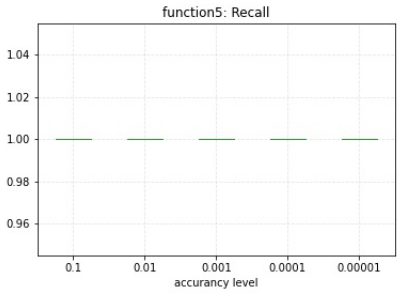

图 24. Function 5: Recall

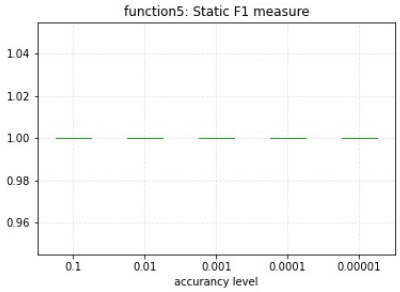

图 25. Function 5: Static F1 score

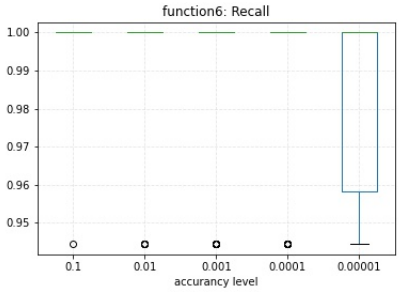

图 26. Function 6: Recall

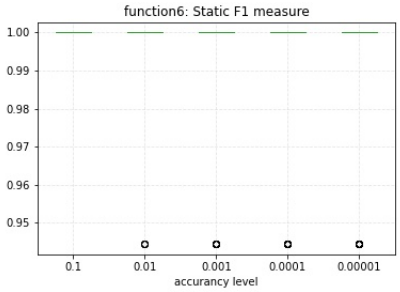

图 27. Function 6: Static F1 score

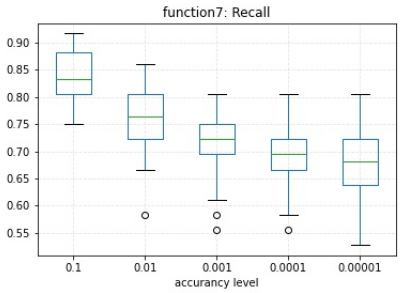

图 28. Function 7: Recall

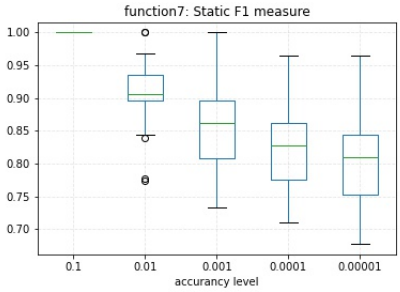

图 29. Function 7: Static F1 score

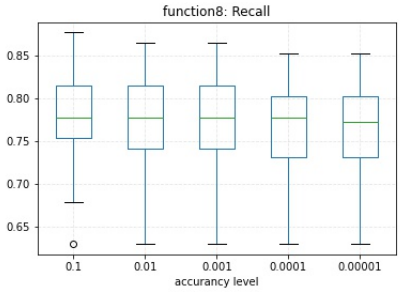

图 30. Function 8: Recall

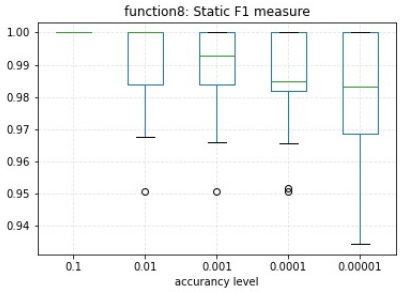

图 31. Function 8: Static F1 score

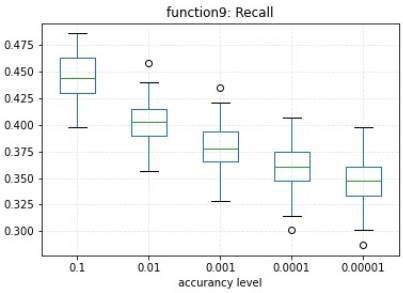

图 32. Function 9: Recall

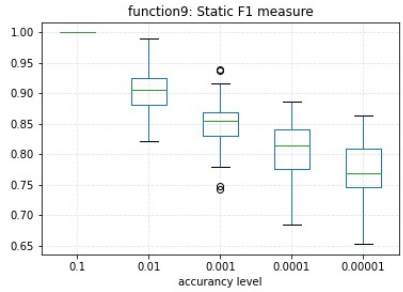

图 33. Function 9: Static F1 score

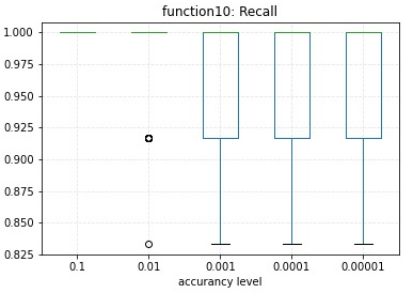

图 34. Function 10: Recall

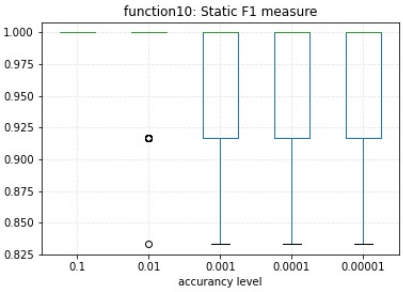

图 35. Function 10: Static F1 score

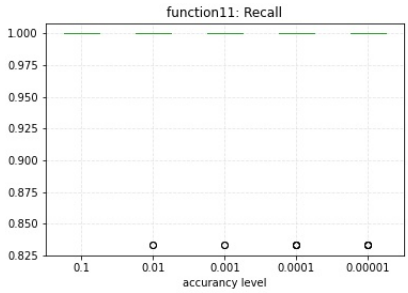

图 36. Function 11: Recall

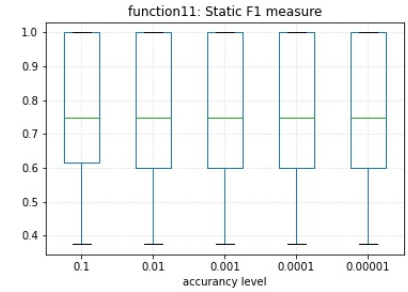

图 37. Function 11: Static F1 score

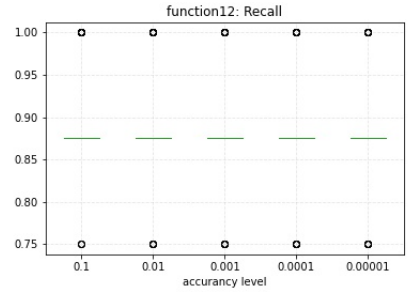

图 38. Function 12: Recall

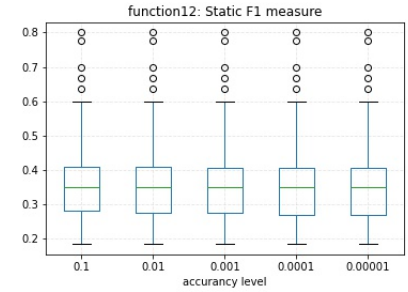

图 39. Function 12: Static F1 score

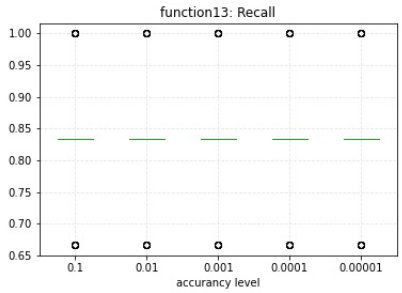

图 40. Function 13: Recall

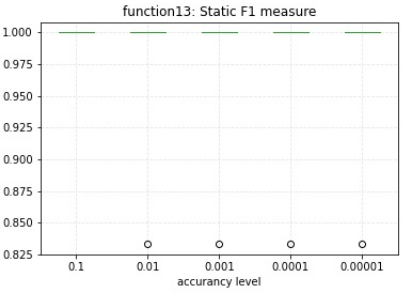

图 41. Function 13: Static F1 score

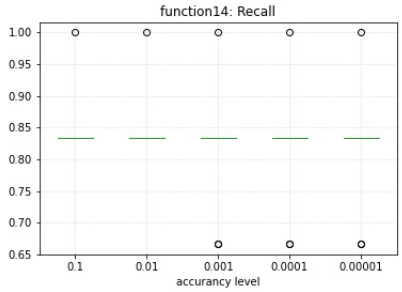

图 42. Function 14: Recall

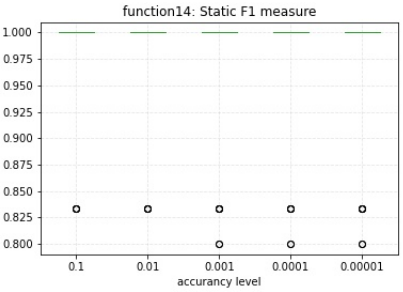

图 43. Function 14: Static F1 score

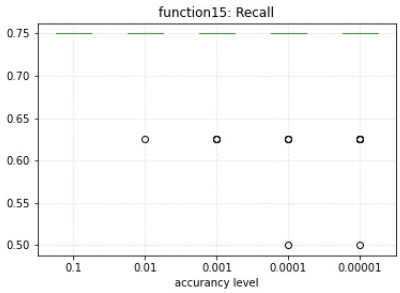

图 44. Function 15: Recall

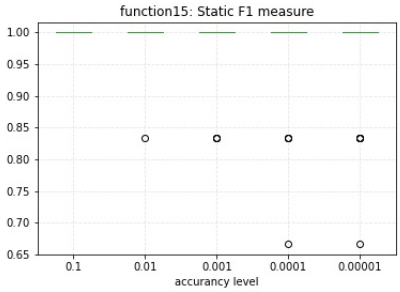

图 45. Function 15: Static F1 score

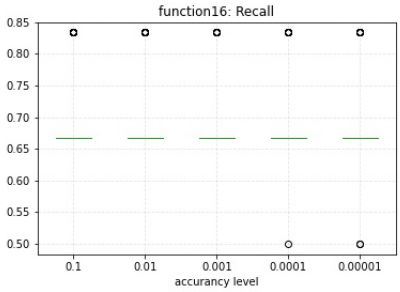

图 46. Function 16: Recall

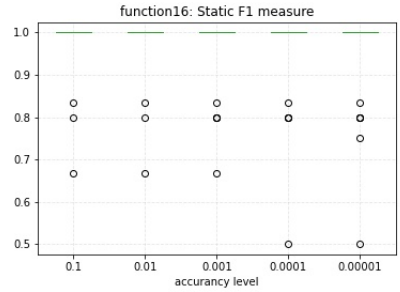

图 47. Function 16: Static F1 score

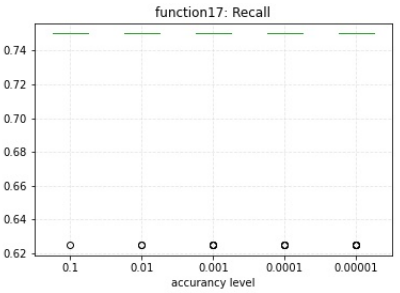

图 48. Function 17: Recall

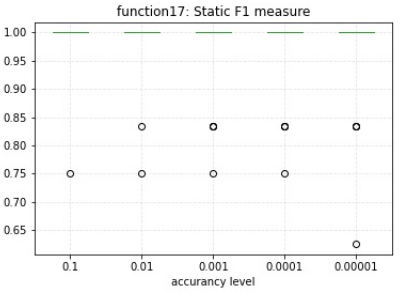

图 49. Function 17: Static F1 score

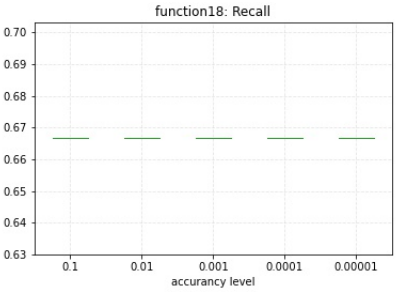

图 50. Function 18: Recall

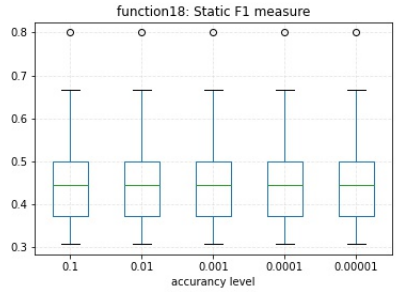

图 51. Function 18: Static F1 score

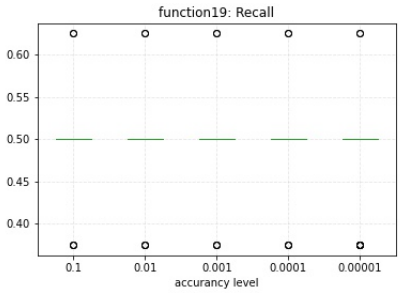

图 52. Function 19: Recall

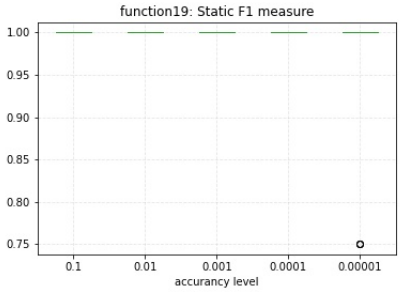

图 53. Function 19: Static F1 score

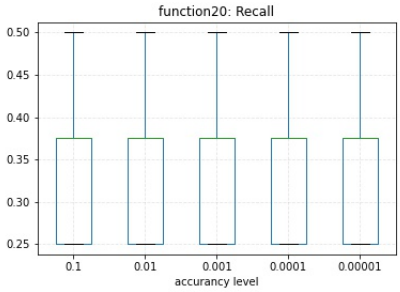

图 54. Function 20: Recall

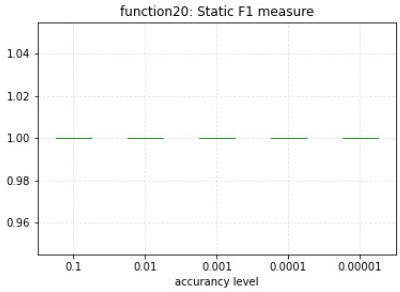

图 55. Function 20: Static F1 score

IV. 总结和未来的工作

本文首先介绍了多波峰优化和 Niching 的基本概念，并分析了使用到的 benchmark 函数的特点，接着详细介绍了本文的算法，即带禁忌区域的自适应协方差演化策略，算法的核心思想是利用 CMSA-ES 去更新每个 Niche，并使用禁忌区域来保证 Exploitation 和 Exploration。最后，根据 benchmark 的特点分三组实验进行，实验结果表明：对于低维度或者较少波峰的函数，算法的基本能搜索到所有波峰，而随着波峰数目或者维度的增加，由于局部搜索能力的限制，算法的 PR 值有所下降。特别的，难度最大的两个函数为 function 9 和 function 20，218 个波峰的 function 9 上 PR 值为 0.4，在 20 维的 function 20 上 PR 为 0.35。

分析实验结论可以得出，本文算法仍然存在以下问题可以优化：

- Exploitation 部分：对于多波峰的情形（如 function 9 的 218 个），算法的局部搜索能力有待优化，可以尝试在每个 Niche 搜索的过程中不断细化搜索的粒度，比如逐代增加子代数量，或者随着收敛速度的降低收缩搜索空间。
- Exploration 部分：对于多维度的情形（如 function 20 的 20 维），算法的问题不是无法收敛，而是多样性不足，有些最优策略并不在 Niche 的搜索空间中，这是由于每个 Niche 的搜索范围是由拒绝半径形成的圆，这个圆无法实际地适应真实 basin 的形状，因此对于拒绝半径的度量，可以采用其他度量方式，如马氏距离等，亦可以通过协方差形成等高线，让 Niche 的搜索空间更加适应实际的 basin。

参考文献

[1] Benchmark Functions for CEC'2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization. 2014.

[2] Ahrari A , Deb K , Preuss M . Multimodal Optimization by Covariance Matrix Self-Adaptation Evolution Strategy with Repelling Subpopulations[J]. Evolutionary Computation, 2016, 25(3):439-471.

[3] http://epitropakis.co.uk/gecco2021/

[4] https://www.zhihu.com/question/318429550

[5] https://zhuanlan.zhihu.com/p/150946035