# IFEP 算法收敛速度与局部搜索能力的优化

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—有界约束的单目标数值优化问题是演化计算领域的经典问题，目前已有的 CEP、FEP、IFEP 以及相关的算法被应用到该问题上并取得一定效果，但是这些算法还存在很大的优化空间，一是收敛速度有待提高，二是需要提高局部搜索能力。针对这两个问题，本文在 IFEP 算法的基础上提出一种优化算法。主要有三个 contributions，一是引入 Simple arithmetic recombination，二是引入精英策略的全局局部搜索，三是将优化的突变标准差系数方程引入有条件局部搜索中。在同样的实验条件下，经过 Wilcoxon signed-rank test 的统计测试表明，该算法比 IFEP 算法拥有更好的收敛速度和更优的最优值。**

***Index Terms*—演化计算，有界约束单目标数值优化，IFEP, 有条件局部搜索，全局局部搜索，交叉重组，精英策略**

## I. 介绍

本文将在 Section I 中介绍所使用到的有界单目标数值优化函数的特点，分析现有算法存在的两大可优化问题；在 Section II 中，本文将针对两大问题讨论解决思路、解决过程，并给出最终算法；在 Section III 中，本文将用可视化和统计两种方式比较 3 种已有算法和 4 种测试算法的性能，并给出相应分析；最后，Section IV 中将总结本文工作，并提出的解决目前算法存在问题的可能方案以及未来工作。

### *A. 有界约束单目标数值优化的 benchmarks*

针对有界约束的带目标数值优化问题，论文 [1] 中使用到了一系列的 benchmark function，为了比较算法性能，本文使用到了该论文中的 function 1-10、function 17 和 function 21，并重新排列了 function 17 和 function 21 的次序为 function 11 和 function 13，其余顺序相同（后续提及的函数编号顺序皆为本文的编号）。这些 functions 都是用于求最小值的问题，但在问题维度和波峰（即极值）上有不同的特点。

从问题维度看，function 1 到 function 10 是高维问题，function 11 和 function 13 则是低维问题。从波峰的角度看，function 1 到 function 7 是单波峰函数，其中 function 6 是 step 函数并且非连续，function 7 是个四次噪声函数，function 8 到 function 13 是多波峰函数，其中 function 8 到 function 10 的局部极值的数量随着维度增加而指数增加，function 11 到 function 13 则只有少数的局部极值。因此，本文所使用到的 benchmark function 可以分为三类：

- 类 1（function 1-7）：高维度、单波峰函数；
- 类 2（function 8-10）：高维度、多波峰函数；
- 类 3（function 11、13）：低维度、少数波峰函数。

function 1 至 function 11 以及 funtion 13 如图 1 至图 12 所示。

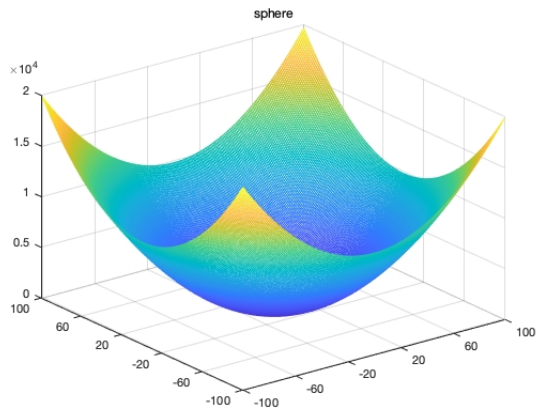


图 1. sphere.

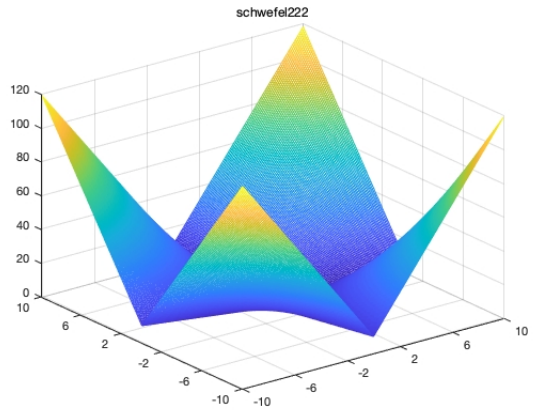


图 2. schwefel222.

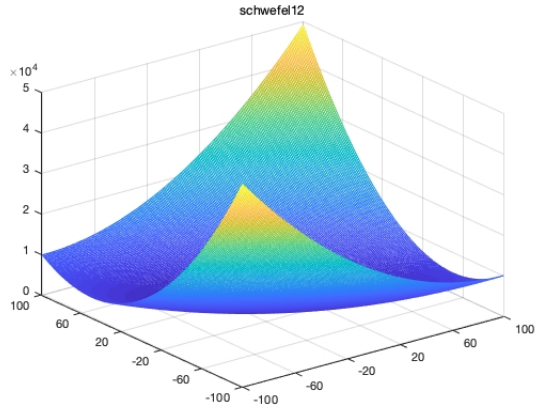


图 3. schwefel12.

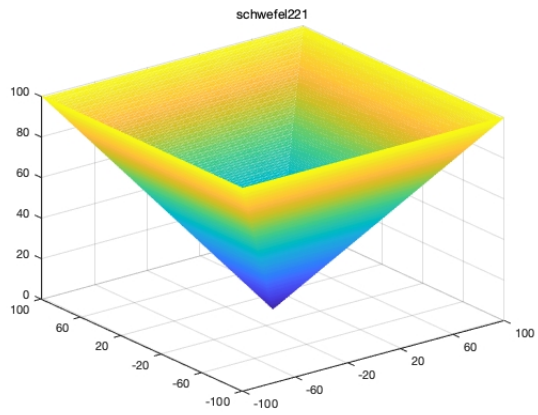


图 4. schwefel221.

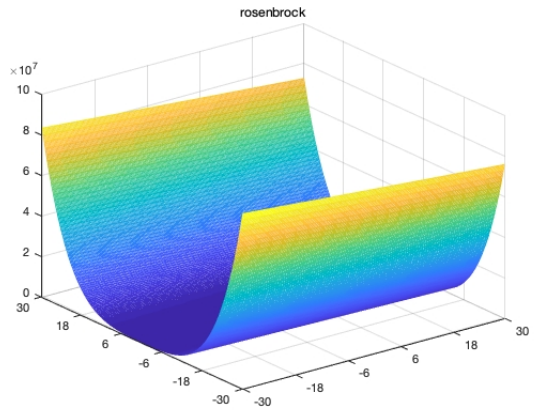


图 5. rosenbrock.

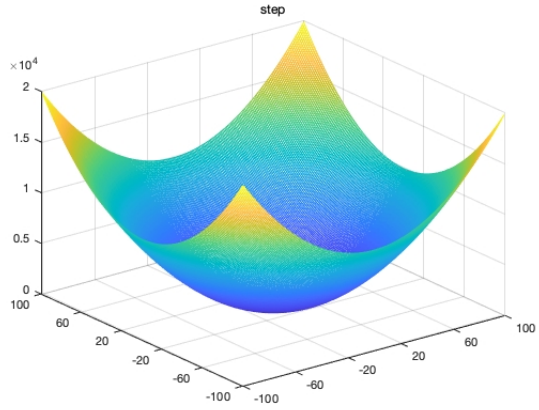


图 6. step.

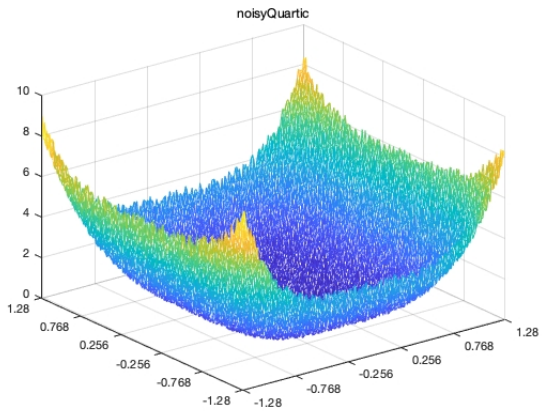


图 7. noisyQuartic.

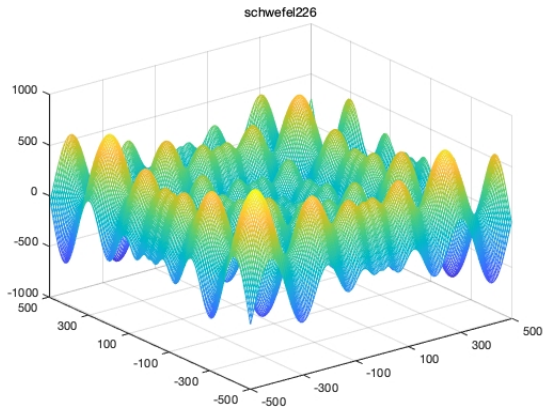


图 8. schwefel226.

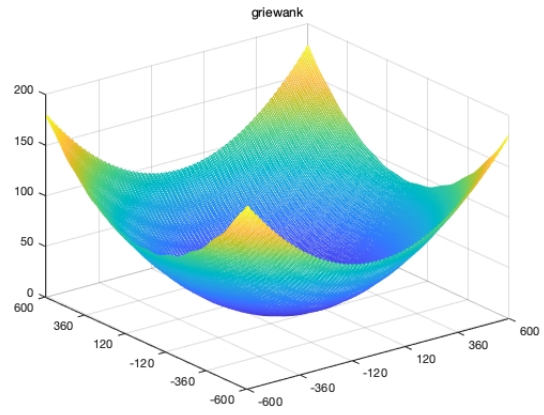


图 11. griewank.

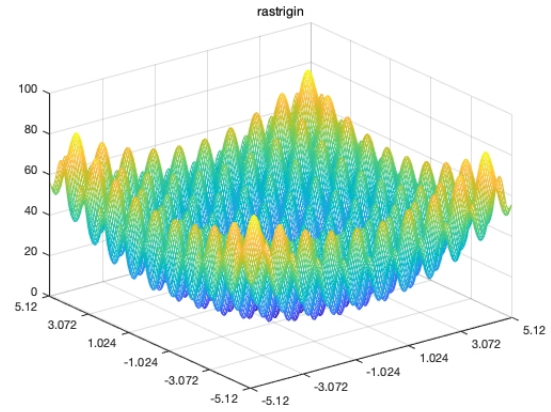


图 9. rastrigin.

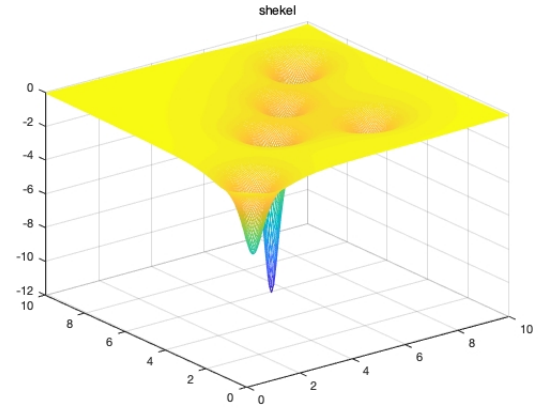


图 12. shekel.

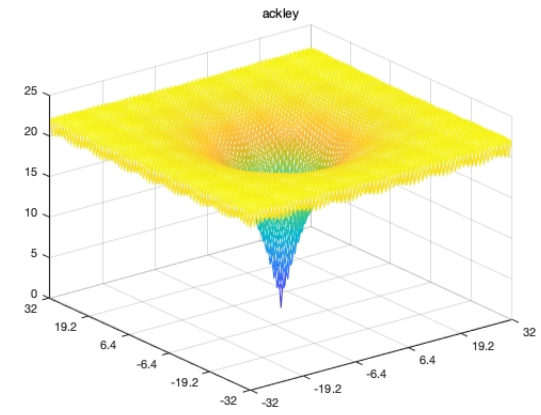


图 10. ackley.

### *B.* 现有算法分析

CEP 算法使用 Gaossian 突变，利用了其 step size 短的特点，具有较好的局部搜索能力。FEP 算法使用 Cauchy 突变，Cauchy 概率密度分布函数的两个尾部平缓，step size 较长，突变时能够产生 long jump，因此比 CEP 算法更好的克服 local optima。而论文 [1] 提出的 IFEP 算法是 FEP 算法的改进，将 Gaossian 突变和 Cauchy 突变结合，利用两种突变分别产生与总群大小一致的两种后代，再于其中选择下一代，达到较好的效果。但是，其**收敛速度**和**最优值的局部搜索**都有很大的提高空间 [1]。

## II. 提出的算法

### A. 算法描述

本文测试了比较了多种算法，并在 IFEP 的基础设计了 4 个尝试改进的算法，包括：

（1）Algorithm 4：在 IFEP 基础上加入 Single arithmetic recombination；

（2）Algorithm 5：在 IFEP 基础上加入 Simple arithemetic recombination；

（3）Algorithm 6：IFEP2；

（4）Algorithm 7：在 Algorithm 5 基础上引入精英策略和带有优化系数方程的 Local search；

以下介绍设计这些算法的考虑。针对以上提出的**两大优化方向**，本文首先考虑**收敛速度**，将收敛速度分为未达到最优值附近的收敛前期和达到最优值附近的收敛后期。收敛前期，提高收敛速度的方式之一是在不过大增加大量搜索计算成本的情况下，扩大搜索空间，提高搜索到最优值附近的可能性，增大跳出局部最优的可能性因此考虑加入 **Recombination**（Crossover）。因为算法的 representation 是 real-value 的，我们选用能够改变实际值的 Arithmetic recombination，设计并测试了 Single 和 Simple 两种算法, 测试结果在 Section V 中给出，结果表明加入 Recombination 的算法在收敛前期表现比未加入 Recombination 的算法较优。而 IFEP2 则是将每个 Population 按照 fitness 排序，fitness 值高的一半个体在下一步使用 Gaossian 突变，fitness 低的个体在下一步使用 Cauchy 突变，测试结果在 Section V 中给出，结果表明 IFEP2 算法收敛慢最优值表现不佳。

在以上测试算法的基础上，本文提出了最终的算法 Algorithm 7。该算法在 Algorithm 5（即 IFEP-Simple）的基础上优化了**最优值附近的局部搜索能力**，主要有两大优化，包括全局的局部搜索和有条件的局部搜索。

**一是引入全局精英局部搜索策略**：在整个演化过程中保持对精英个体的搜索，将 20 个最优个体筛选出来，进行小方差的高斯突变并产生 20 个后代，将后代加入下一代筛选的范围中。

**二是在有条件局部搜索中引入优化的突变标准差系数方程**：定义进入 Local search 的阀值为：

$$LocalSearchThreshold = \frac{1}{(ub - lb) * degree(\mu)} \quad (1)$$

其中 $\mu$ 为种群个体数，$degree(\mu)$ 代表取与 $\mu$ 相同 10 幂次级的数，例如 $\mu$ 为 50，则 $degree(\mu)$ 为 10，$ub$ 为 benchmark function 的上界，$lb$ 为下界。定义两代种群之间的演变倾向与程度为：

$$SearchGap = averageFitness_{current} - averageFitness_{last} \quad (2)$$

其中 $averageFitness_{current}$ 是当前种群所有个体 fitness 的平均值，$averageFitness_{current}$ 是上一代种群所有个体 fitness 的平均值,，SearchGap 大于 0 表示种群整体进化。

规定当满足以下 **2 个条件时**，进入局部搜索。

（1）$|SearchGap| < LocalSearchThreshold$

（2）$SearchGap > 0$

局部搜索中进行了 **2 项操作**，一是取消交叉重组，二是优化突变所使用到的突变标准差系数方程 $\eta$ 为：

$$\eta_i'(j) = \begin{cases} \frac{\eta_i(j) exp(\tau' N(0,1) + \tau N_j(0,1))}{nbGeneration} & localsearch \\ \eta_i(j) exp(\tau' N(0,1) + \tau N_j(0,1)) & otherwise \end{cases} \quad (3)$$

其中 nbGeneration 是种群代数，初始化为第一代。$\tau$ 论文 [1] 中定量分析出当 current point 在 optimal point 附近时 step size 大于一定值时，越大则到达 optimal point 的概率越低，因此此时要降低 step size。以上 $\eta$ 改进公式完成的即是这个功能，该功能与直接改变 Cauchy 步长效果相同。本算法的伪代码如下：

**Input:** benchmark 名称, benchmark 维度, benchmark 下界，benchmark 上界，评估次数

**Output:** 每代最优 fitness，每代平均 fitness，目前的最优策略

1: 参数初始化（种群大小 $\mu = 50$），使用均匀分布初始化种群，并评价种群 fitness；
2: **while** $nEval < nEvaluationS$ **do**
3:  **if** $|SearchGap| > LocalSearchThreshold$ **then**
4:  针对 Gaossian 突变和 Cauchy 突变所使用到的父代分别做 Simple arithmetic recobination 得到两个大小都为 $\mu$ 的种群 OffspringGaossian 和 OffspringCauchy；
5:  **end if**

6: **if** $|SearchGap| < LocalSearchThreshold$ and $SearchGap > 0$ **then**

7: 设置 $\eta_i'(j) = \frac{\eta_i(j)exp(\tau' N(0,1)+\tau N_j(0,1))}{nbGeneration}$;

8: **end if**

9: 在 OffspringGaossian 和 OffspringCauchy 上分别进行高斯突变和柯西突变，得到总大小为 $2\mu$ 的 Offspring。将 Population 和 Offspring 都加入到总总群 AllIndividuals 中;

10: 取 Population 中 fitness 最高的前 20 个，进行更小范围的高斯突变，得到 20 个个体并加入总种群 AllIndividuals 中，此时 AllIndividuals 个体数量为 $3\mu + 20$;

11: 通过 Tournament Selection 的方式随机在 AllIndividuals 中选出 10 个用于比较的个体，将 AllIndividuals 中所有个体与比较个体比较 fitness，选出 $\mu$ 个 fitness 最高的个体作为下一代;

12: 更新 Population 等，记录数据;

13: **end while**

14: 返回每代最优 fitness，每代平均 fitness，目前的最优策略;

## III. 实验结果与讨论

### A. 实验设置

本文的所有实验中，每个算法都在 12 个 benchmark 上进行 30 次的独立重复实验，实验参数设置为：Population size 为 100，Tournament size 为 10。

在每个 benchmark 上，每个算法对个体进行 Evaluation 的总次数相同。

在每个算法中，种群初始化方式相同，取值是在取值区间上均匀分布的随机数，高斯突变和柯西突变的参数 $\eta$ 初始值都为 3，Simple arithmetic recombination 和 Single arithmetic recombination 的交叉权值参数 $\alpha$ 为 0.5。

本文使用到的统计测试方法为 **Wilcoxon signed-rank test**，假设的 $H_0$ 两组数据的差异遵循围绕零的对称分布，即两组数据无显著差异，其中显著性水平都设置为 0.05。

### B. 实验内容和分析

本文共仿真测试了 7 个算法，包括 3 个现有算法 [1], 以及针对这三个算法存在的共同问题而设计的 4 个改进测试算法，它们分别是：

- Algorithm 1: Classical EP;
- Algorithm 2: Fast EP;
- Algorithm 3: Improved FEP;
- Algorithm 4: IFEP with Single arithmetic recombination;
- Algorithm 5: IFEP with Simple arithmetic recombination;
- Algorithm 6: IFEP2;
- Algorithm 7: Algorithm 5 with Local Search and Elitist strategy;

**实验记录图表介绍：**

**图：** 图 13 到图 18 为 Algorithm 3、4、5 在 function 4、10、13 上的收敛曲线。图中 x 轴代表 Evaluation 次数，y 轴代表当前总群中 function 的最优值 recordedBestY 或者平均值 recordedAvgY。图 19 到图 42 为 Algorithm 3 和 Algorithm 7 在 function 1-12 以及 function 13 下的收敛曲线。

**表：** 表 1 至表 12 分别用于比较 7 种算法在不同 benchmark function 下的平均最优值和标准差。表中 F 代表 function 的序号，Algo 代表 Algorithm 的序号，h 值表示进行 Wilcoxon signed-rank test 的结果，h = 0 表示接受 $H_0$ 假设，即两组数据无明显差别，h = 1 表示拒绝假设 $H_0$，即两组数据有明显差别。rank 表示 Wilcoxon signed-rank test 的总秩，值越大表示该算法最优值优于基准算法的程度越高。win 的值表示该算法在此 function 上与基准算法（用于比较的算法）的最优值比较结果，win = 1 表示优于基准算法（在本文中即函数值小于），win = -1 表示差于基准算法，win = 0 表示相同。在每个表格中，基准算法的 h 值、rank 值、win 值都置空以作标识。

*1)* ***实验 1:*** 对比加入 Recombination 的 IFEP 与传统 IFEP 的收敛速度，即将 Algorithm 4 和 Algorithm 5 与 Algorithm 3 进行收敛速度比较。以上三个算法在 12 个 benchmark function 上进行测试。选取第 1 类函数中的 function 4、第 2 类函数中的 function 10 和第 3 类函数中的 function 13 的 recordedBestY 和 recordedAvg 收敛曲线来分析，如图 13 至图 18 所示。

- 在 function 4 上，带有交叉重组的算法 Algorithm4 和 Algorithm 5 在 evauation 的中后期收敛速度明显快于 Algorithm 3;

- 在 function 10 上，使用 Single arithmetic recombination 算法 Algorithm 4 收敛速度快于 Algorithm3；
- 在 function 10 上，使用 Simple arithmetic recombination 算法 Algorithm 5 搜索空间大于 Algorithm 3、4，因此不容易陷入局部最优。

综合表 1 至表 12，可得到结论：尽管带有 Recombination 的算法在最优值上并没有表现得明显优于 IFEP 算法，但是具有不易陷入局部最优、收敛速度较快的特点。

而针对 Recombination 的对比选择，Simple 和 Single 在本次实验中收敛速度和最优值都没有表现出明显的性能差别，两者在理论上的区别在于交叉的比例，Single 选取单个 point 上的 value 进行交叉，而 Simple 随机选取一个 point，将该点后的每个 value 都进行交叉，改变的搜索空间大于 Single 的概率较高，陷入局部最优的可能性更低，因此本文在最终算法中选取了 Simple 算法作为 Recombination 的方法。

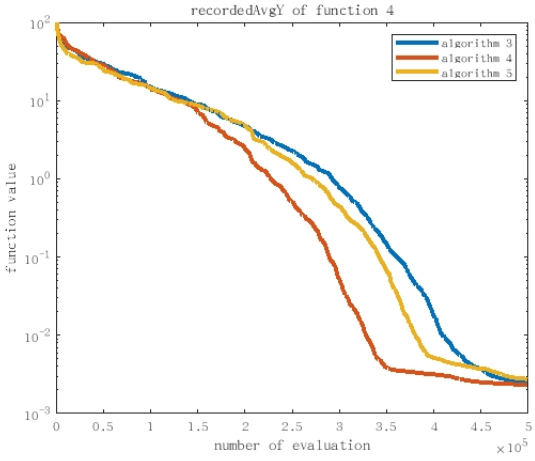


图 13. function4-recordedAvgY.

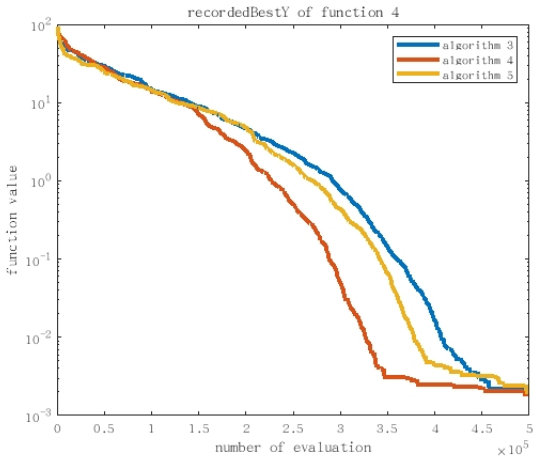


图 14. function4-recordedBestY.

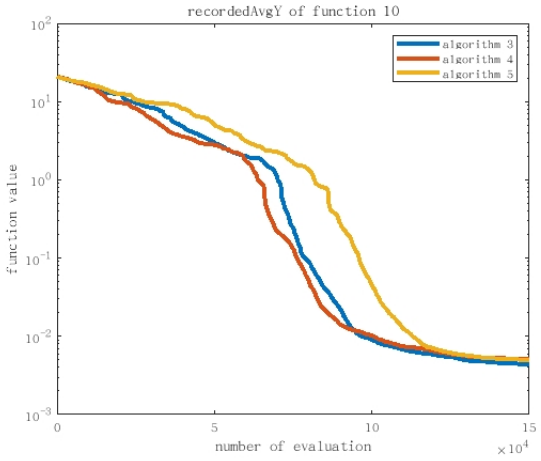


图 15. function10-recordedAvgY.

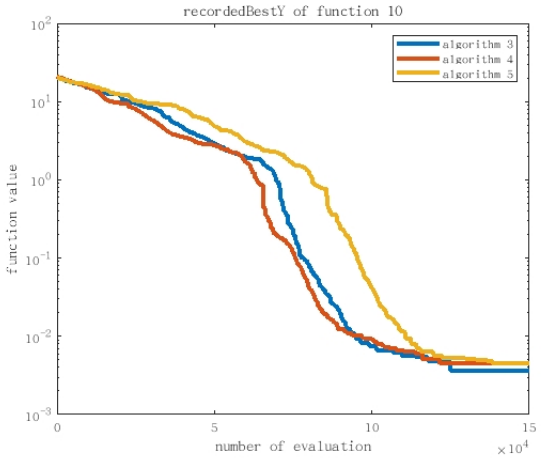


图 16. function10-recordedBestY.

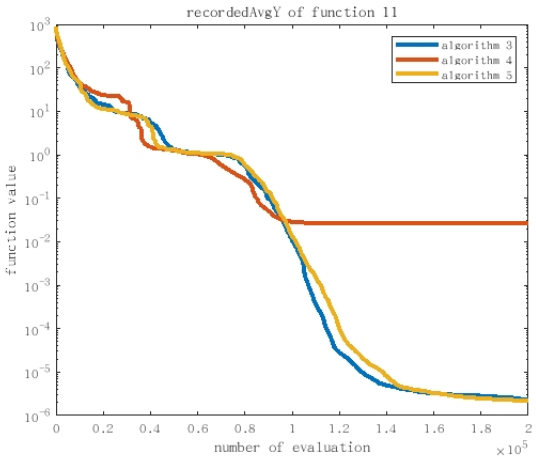


图 17. function11-recordedAvgY.

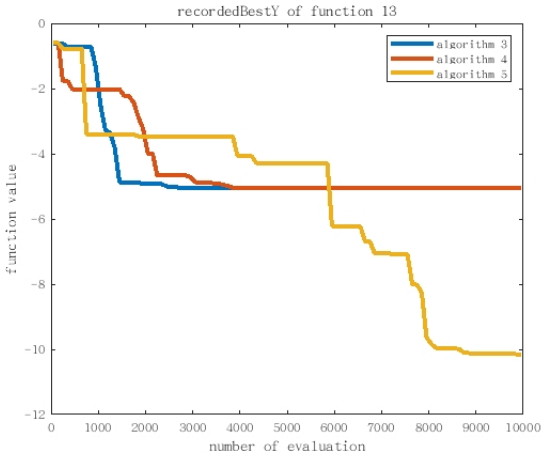


图 18. function13-recordedBestY.

*2)* **实验 *2*::** 对比加入 Local search 的 Algorithm 7 和使用 IFEP 的 Algorithm 3 的局部搜索能力和收敛速度。以上 2 个算法在 12 个 benchmark function 上进行测试.

首先分析**局部搜索能力**，如表 1 至表 12 所示，除了 function 5 和 9，Algorithm 7 的 Mean best 值在 10 个 function 上都比 Algorithm 3 低，且是 4 个算法中的最低值，除此之外，符号秩检验的总秩值也较高。这表明 Algorithm 7 所用到的局部搜索策略较好地起到了增强局部搜索能力的作用。

其次分析**收敛速度**，如图 19 至图 42 所示，Algorithm 7 在 function 1、3、4、6、7、10、13 共 7 个 function 上收敛速度明显快于 Algorithm 3，其余 5 个 function 上收敛速度相近。这表明 Algorithm 7 能够一定程度上优化收敛速度。

表 I
对比不同算法在 FUNTION 1 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 1 | 3 | n | n | | 3.850567e-05 | 5.320645e-06 |
| | 4 | 1 | 155 | 1 | 3.603014e-05 | 5.572460e-06 |
| | 5 | 1 | 137 | 1 | 3.666991e-05 | 4.234818e-06 |
| | 7 | 1 | 465 | 1 | **1.156249e-08** | 2.566840e-08 |

表 II
对比不同算法在 FUNTION 2 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 2 | 3 | n | n | | 2.239604e-02 | 2.180279e-03 |
| | 4 | 1 | -109 | -1 | 2.294617e-02 | 1.529198e-03 |
| | 5 | 1 | -81 | -1 | 2.276983e-02 | 1.738601e-03 |
| | 7 | 1 | 283 | 1 | **2.070666e-02** | 2.412219e-03 |

表 III
对比不同算法在 FUNTION 3 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 3 | 3 | n | n | | 2.557747e-04 | 1.110279e-04 |
| | 4 | 1 | -39 | -1 | 2.735368e-04 | 1.452097e-04 |
| | 5 | 1 | -117 | -1 | 3.004452e-04 | 1.743555e-04 |
| | 7 | 1 | 129 | 1 | **2.100748e-04** | 6.796215e-05 |

表 IV
对比不同算法在 FUNTION 4 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 4 | 3 | n | n | | 2.266498e-03 | 2.738139e-04 |
| | 4 | 1 | 21 | 1 | 2.260384e-03 | 3.007821e-04 |
| | 5 | 1 | 49 | 1 | 2.249820e-03 | 3.076223e-04 |
| | 7 | 1 | 465 | 1 | **1.606666e-03** | 2.210133e-04 |

表 V
对比不同算法在 FUNTION 5 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 5 | 3 | n | n | | 7.899273e+00 | 1.488977e+01 |
| | 4 | 1 | -87 | -1 | 8.799457e+00 | 1.404476e+01 |
| | 5 | 1 | 19 | 1 | **5.278096e+00** | 5.330641e+00 |
| | 7 | 1 | -183 | -1 | 1.282135e+1 | 1.817708e+1 |

表 VI
对比不同算法在 FUNTION 6 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 6 | 3 | n | n | | 3.819689e-05 | 6.611472e-06 |
| | 4 | 1 | -39 | -1 | 3.874954e-05 | 6.544571e-06 |
| | 5 | 1 | 133 | 1 | 3.566280e-05 | 5.871946e-06 |
| | 7 | 1 | 465 | 1 | **7.178198e-9** | 1.567754e-08 |

表 VII
对比不同算法在 FUNTION 7 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 7 | 3 | n | n | | 1.163441e+02 | 2.125034e+01 |
| | 4 | 1 | 35 | 1 | 1.102002e+02 | 2.901068e+01 |
| | 5 | 1 | -37 | -1 | 1.161152e+02 | 2.326247e+01 |
| | 7 | 1 | 465 | 1 | **2.694169e-02** | 8.296765e-03 |

表 VIII
对比不同算法在 FUNTION 8 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 8 | 3 | n | n | | -1.105997e+04 | 2.284901e+02 |
| | 4 | 1 | -59 | -1 | -1.099830e+04 | 3.765953e+02 |
| | 5 | 1 | -129 | -1 | -1.092814e+04 | 4.070844e+02 |
| | 7 | 1 | 59 | 1 | **-1.110368e+04** | 3.567361e+02 |

表 IX
对比不同算法在 FUNTION 9 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 9 | 3 | n | n | | 5.070210e-03 | 1.235013e-03 |
| | 4 | 1 | -163 | -1 | **3.891059e-02** | 1.819514e-01 |
| | 5 | 1 | -73 | -1 | 7.157849e-02 | 2.522047e-01 |
| | 7 | 1 | -21 | -1 | 3.016201e-01 | 5.918225e-01 |

表 X
对比不同算法在 FUNTION 10 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 10 | 3 | n | n | | 4.401756e-03 | 4.660273e-04 |
| | 4 | 1 | -143 | -1 | 4.568987e-03 | 3.375865e-04 |
| | 5 | 1 | -211 | -1 | 4.677204e-03 | 4.840279e-04 |
| | 7 | 1 | 359 | 1 | **3.694432e-03** | 9.535198e-04 |

表 XI
对比不同算法在 FUNTION 11 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 11 | 3 | n | n | | 5.420949e-02 | 5.894522e-02 |
| | 4 | 1 | 51 | 1 | 4.659441e-02 | 5.392762e-02 |
| | 5 | 1 | 101 | 1 | 3.826154e-02 | 4.871051e-02 |
| | 7 | 1 | 319 | 1 | **9.845995e-03** | 1.265269e-02 |

表 XII
对比不同算法在 FUNTION 13 上的性能

| F | Algo | h | rank | win | Mean best | Std Dev |
|---|---|---|---|---|---|---|
| 13 | 3 | n | n | | -5.870670e+00 | 2.275363e+00 |
| | 4 | 1 | 57 | 1 | -6.104369e+00 | 2.351020e+00 |
| | 5 | 1 | 53 | 1 | -6.490755e+00 | 2.538487e+00 |
| | 7 | 1 | 211 | 1 | **-7.283242e+00** | 2.795948e+00 |

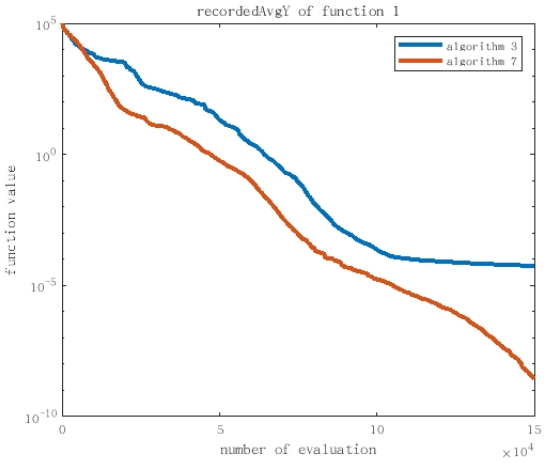


图 19. function1-recordedAvgY.

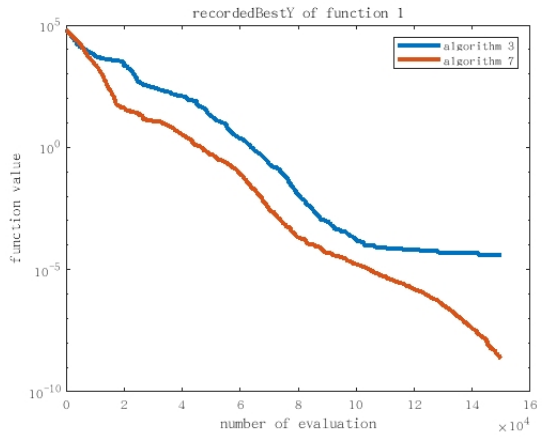


图 20. function1-recordedBestY.

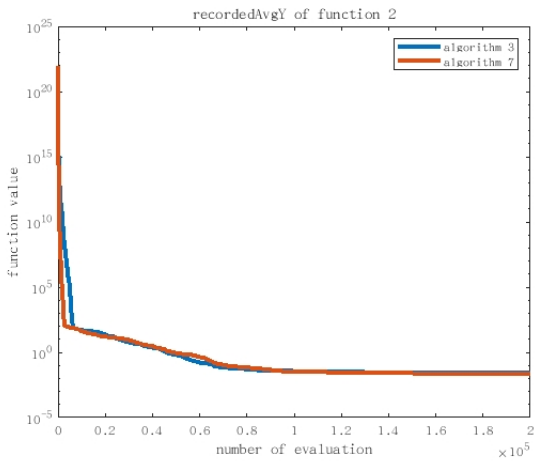


图 21. function2-recordedAvgY.

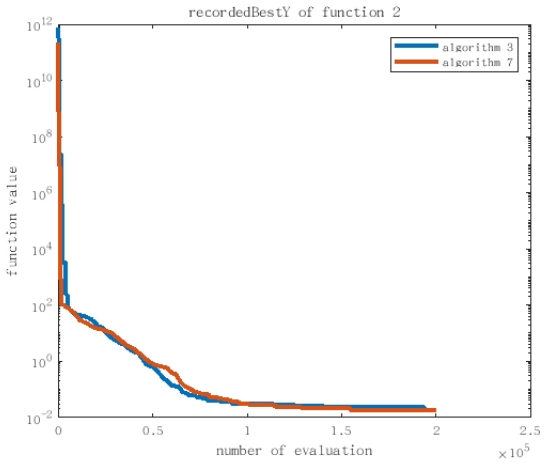


图 22. function2-recordedBestY.

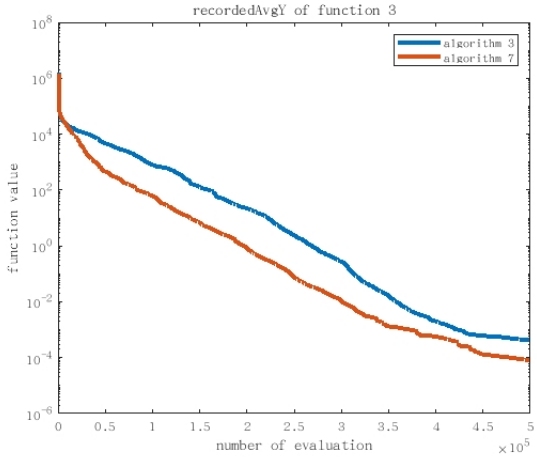


图 23. function3-recordedAvgY.

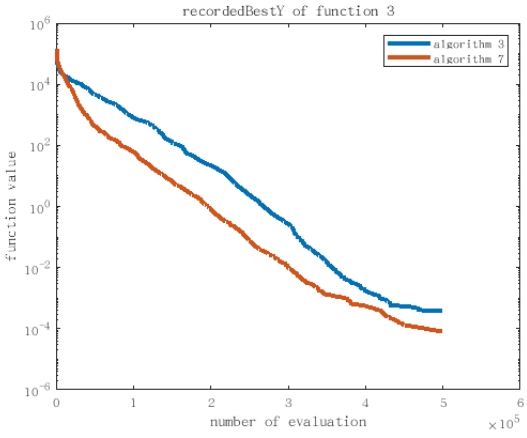


图 24. function3-recordedBestY.

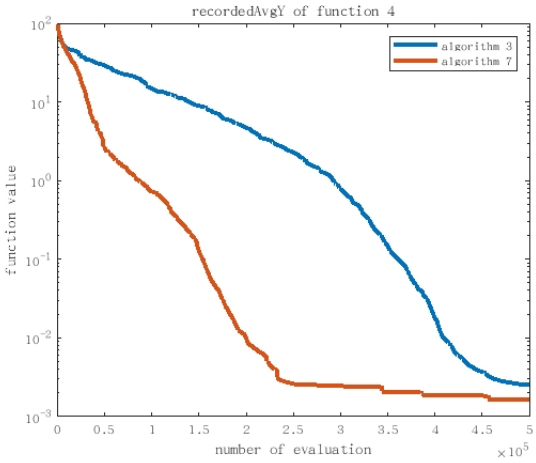


图 25. function4-recordedAvgY.

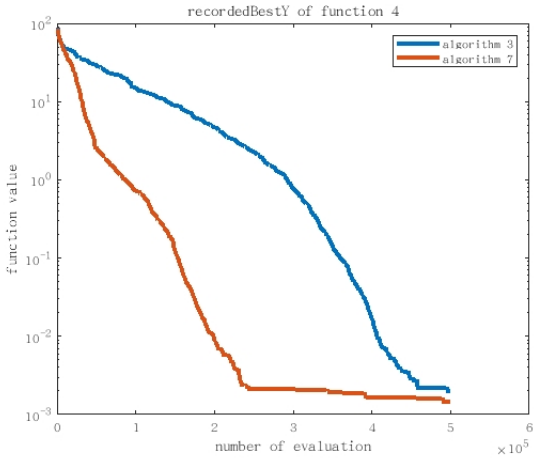


图 26. function4-recordedBestY.

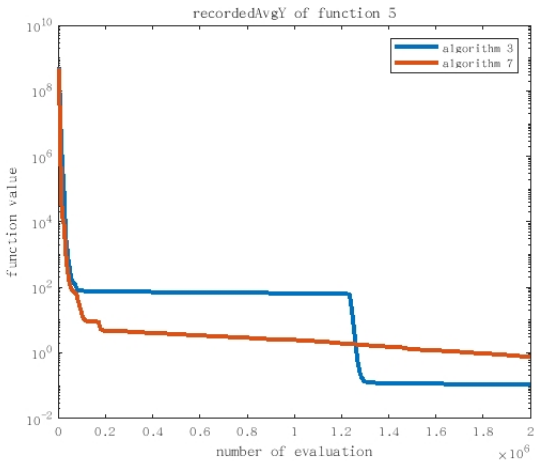


图 27. function5-recordedAvgY.

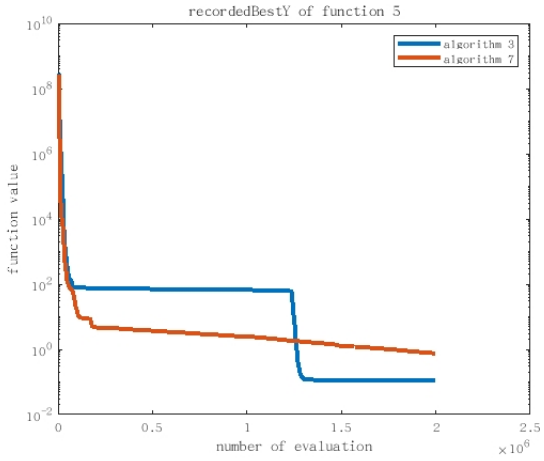


图 28. function5-recordedBestY.

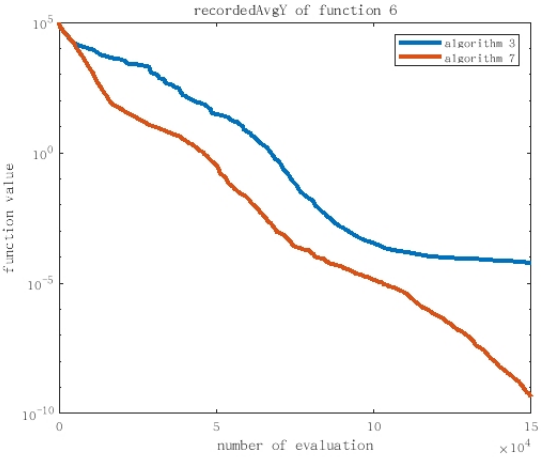


图 29. function6-recordedAvgY.

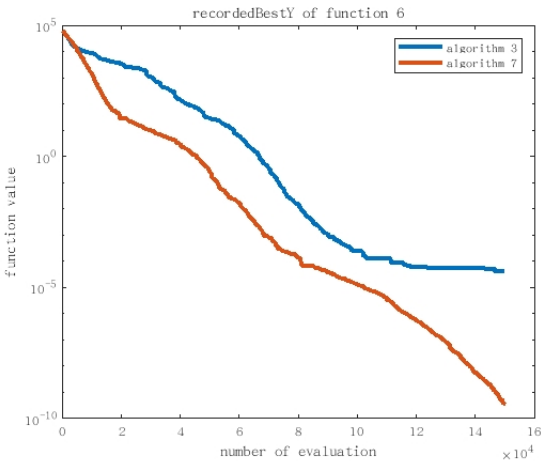


图 30. function6-recordedBestY.

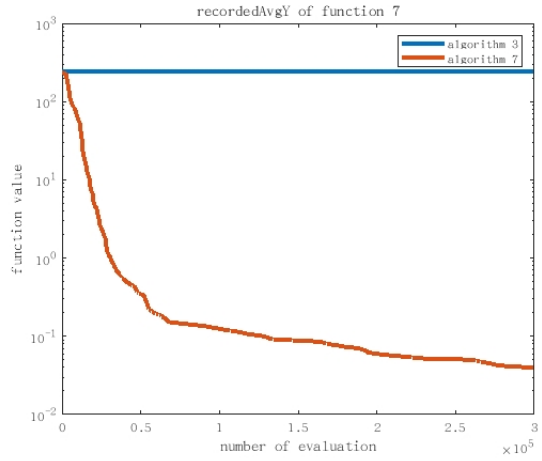


图 31. function7-recordedAvgY.

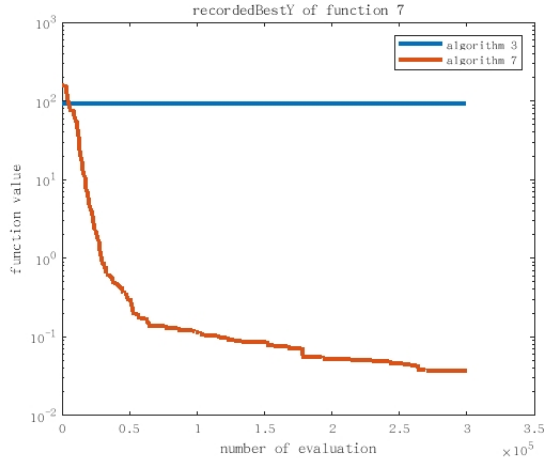


图 32. function7-recordedBestY.

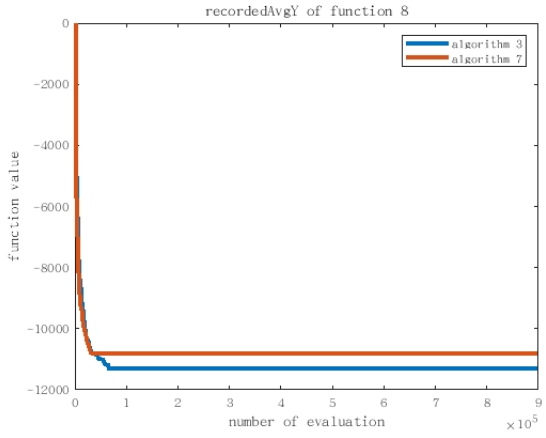


图 33. function8-recordedAvgY.

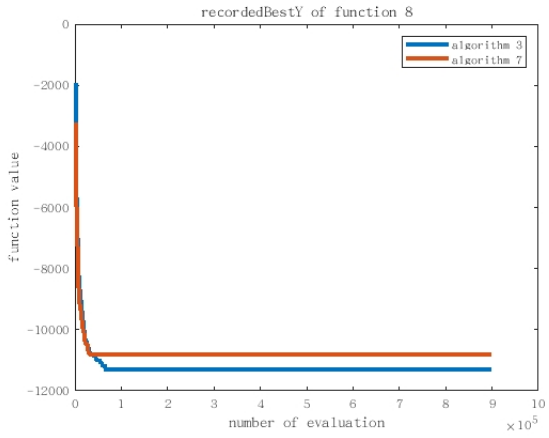


图 34. function8-recordedBestY.

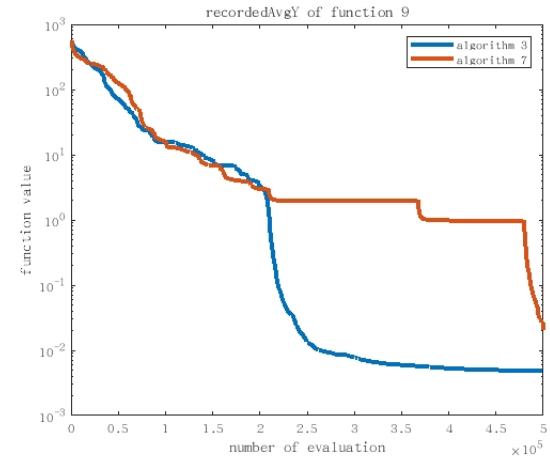


图 35. function9-recordedAvgY.

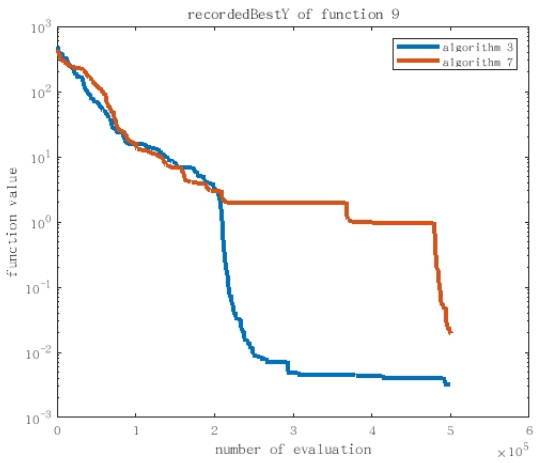


图 36. function9-recordedBestY.

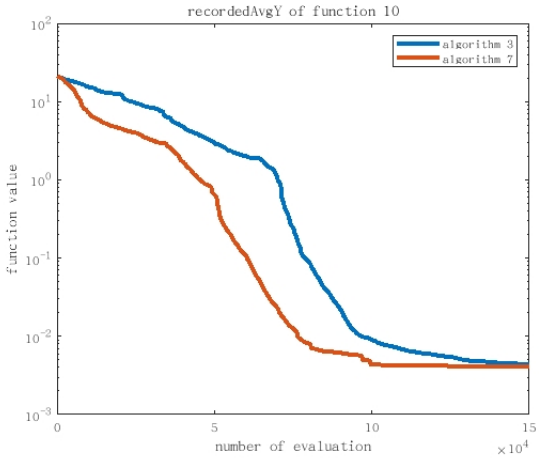


图 37. function10-recordedAvgY.

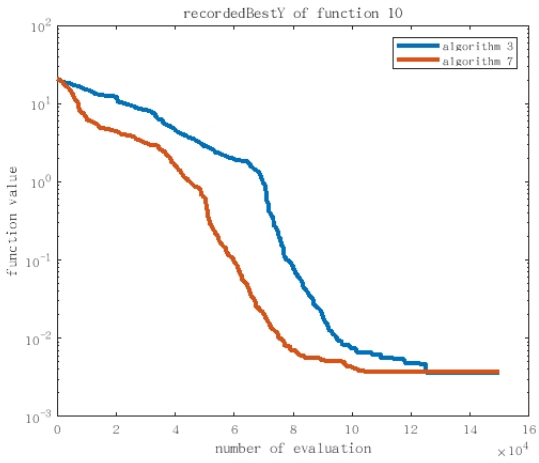


图 38. function10-recordedBestY.

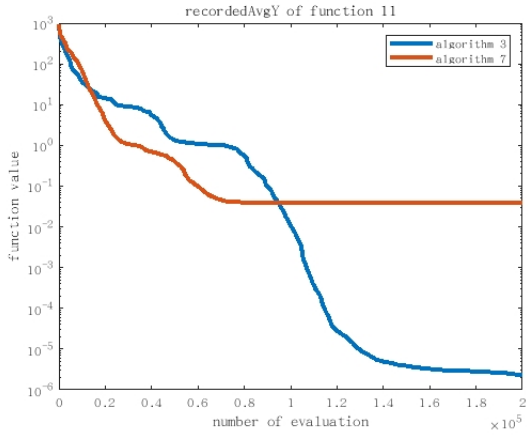


图 39. function11-recordedAvgY.

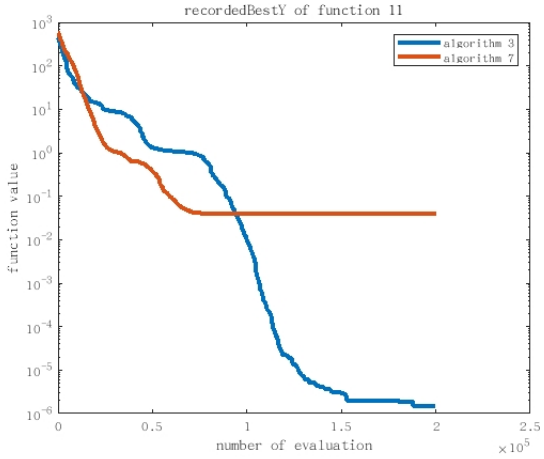


图 40. function11-recordedBestY.

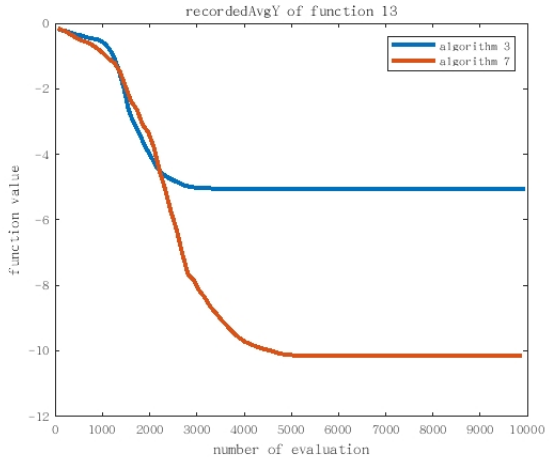


图 41. function13-recordedAvgY.

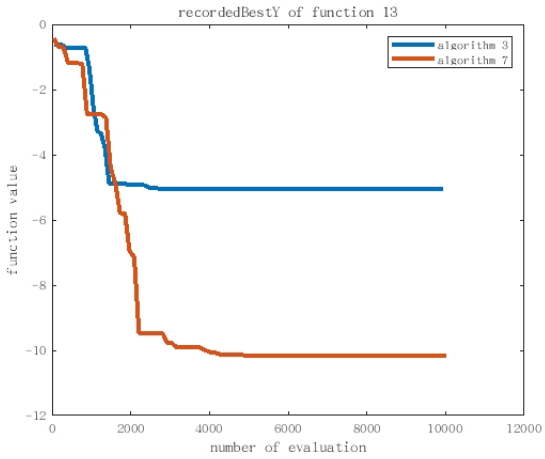


图 42. function13-recordedBestY.

## IV. 总结和未来的工作

本文首先介绍了常用的有界约束的单目标数值优化问题，分析已有算法存在的两大优化空间，即收敛速度和局部搜索能力两大问题。针对这两大问题分两步进行优化，复现了 3 个算法，设计了 4 个优化测试算法，并使用 Wilcoxon signed-rank test 进行统计测试。

实验结果表明，引入 Recombination 能够扩大算法搜索空间，减少进入局部最优的可能性，并一定程度提高整体收敛速度，而本文最终算法 Algorithm 7 所使用的全局精英局部搜索策略以及优化的突变标准差系数方程可以明显加快算法在最优值附近的收敛速度，并获得更优的最优值。

但是本文算法仍然存在以下问题可以优化：

- 本文最终算法所使用的局部搜索阈值公式由种群数量、function 要求的上下界决定，考虑的因素不足以应对所有的局部搜索场合，并且只是一个定性的经验公式，未来将引入更多衡量标准，实现真正的精确动态局部搜索阈值判断。
- 本文在优化算法上缺少足够的定量理论分析，在突变标准差系数方程上还有较大的定量定性优化空间。
- 本文所使用的 Simple arithmetic recombination 算法针对单个每个基因做计算时，实际是起到一个取中间值的作用，类似频域上的一阶低通滤波算法。如果最优值所在点位于因变量取值区间的中间，该算法能够提供较强的收敛速度，但如果最优值所在点远离因变量取值区间的中间（如 function 8 的区

间为 [-500,500]，但最优值所在的点各个维度因变量皆为 420.9687)，则该算法对收敛速度的改善效果将大大降低，并未起到扩大搜索空间的效果，此时，若有先验知识则可以在重新计算每个基因时提供一个偏置值，若无先验知识，则需要采用其他交叉重组算法。

## 参考文献


[1] X. Yao, Y. Liu, and G. Lin. "Evolutionary Programming Made Faster." IEEE Transactions on Evolutionary Computation 3.2 (1999): 82-102.

[2] T. Back, D. B. Fogel, and Z. Michalewicz (eds.), "Handbook of Evolutionary Computation." IOP Publ. Co. and Oxford University Press, 1997.

[3] KH. Liang, X. Yao and C. Newton, "Evolutionary search of approximated N-dimensional landscapes." International Journal of Knowledge-Based Intelligent Engineering Systems, 4(3):172-183, July 2000.

[4] A. E. Eiben and J.E. Smith, "Introduction to Evolutionary Computing." Berlin, Heidelberg: Springer Berlin / Heidelberg, 2015. Natural Computing Ser.


## V. 附录

表 XIII
对比 7 种算法在 FUNCTION 1 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 1.911974e-04 | 3.552178e-04 |
| FEP | 5.749230e-04 | 1.248547e-04 |
| IFEP | 3.850567e-05 | 5.320645e-06 |
| IFEP-Single | 3.603014e-05 | 5.572460e-06 |
| IFEP-Simple | 3.666991e-05 | 4.234818e-06 |
| EIFEP | 4.560514e-05 | 1.557243e-05 |
| IFEP-Simple-Local | 1.156249e-08 | 2.566840e-08 |

表 XIV
对比 7 种算法在 FUNCTION 2 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 2.218611e-02 | 1.491252e-03 |
| FEP | 6.952579e-02 | 5.138582e-03 |
| IFEP | 2.239604e-02 | 2.180279e-03 |
| IFEP-Single | 2.294617e-02 | 1.529198e-03 |
| IFEP-Simple | 2.276983e-02 | 1.738601e-03 |
| EIFEP | 2.480120e-02 | 1.557243e-05 |
| IFEP-Simple-Local | 2.070666e-02 | 2.412219e-03 |

表 XV
对比 7 种算法在 FUNCTION 3 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 8.673721e-02 | 2.071565e-01 |
| FEP | 7.733971e-02 | 8.306234e-02 |
| IFEP | 2.557747e-04 | 1.110279e-04 |
| IFEP-Single | 2.735368e-04 | 1.452097e-04 |
| IFEP-Simple | 3.004452e-04 | 1.743555e-04 |
| EIFEP | 2.644756e-03 | 4.679128e-03 |
| IFEP-Simple-Local | 2.100748e-04 | 6.796215e-05 |

表 XVI
对比 7 种算法在 FUNCTION 4 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 1.417930e+00 | 1.097327e+00 |
| FEP | 4.528365e+01 | 4.722728e+01 |
| IFEP | 2.266498e-03 | 2.738139e-04 |
| IFEP-Single | 2.260384e-03 | 3.007821e-04 |
| IFEP-Simple | 2.249820e-03 | 3.076223e-04 |
| EIFEP | 1.879822e-01 | 2.660528e-01 |
| IFEP-Simple-Local | 1.606666e-03 | 2.210133e-04 |

表 XVII
对比 7 种算法在 FUNCTION 5 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 8.514414e+00 | 1.690491e+01 |
| FEP | 2.612524e+01 | 2.768825e+01 |
| IFEP | 7.899273e+00 | 1.488977e+01 |
| IFEP-Single | 8.799457e+00 | 1.404476e+01 |
| IFEP-Simple | 5.278096e+00 | 5.330641e+00 |
| EIFEP | 1.107272e+01 | 1.916412e+01 |
| IFEP-Simple-Local | 1.282135e+1 | 1.817708e+1 |

表 XVIII
对比 7 种算法在 FUNCTION 6 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 2.839897e-04 | 8.761551e-04 |
| FEP | 5.553128e-04 | 9.587795e-05 |
| IFEP | 3.819689e-05 | 6.611472e-06 |
| IFEP-Single | 3.874954e-05 | 6.544571e-06 |
| IFEP-Simple | 3.566280e-05 | 5.871946e-06 |
| EIFEP | 4.917433e-05 | 2.234761e-05 |
| IFEP-Simple-Local | 7.178198e-9 | 1.567754e-08 |

表 XIX
对比 7 种算法在 FUNCTION 7 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 9.577138e+01 | 1.415748e+01 |
| FEP | 1.498663e-02 | 4.054383e-03 |
| IFEP | 1.163441e+02 | 2.125034e+01 |
| IFEP-Single | 1.102002e+02 | 2.901068e+01 |
| IFEP-Simple | 1.161152e+02 | 2.326247e+01 |
| EIFEP | 1.100350e+02 | 2.167343e+01 |
| IFEP-Simple-Local | 2.694169e-02 | 8.296765e-03 |

表 XX
对比 7 种算法在 FUNCTION 8 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | -7.857957e+03 | 7.564452e+02 |
| FEP | -1.065448e+04 | 2.934297e+02 |
| IFEP | -1.105997e+04 | 2.284901e+02 |
| IFEP-Single | -1.099830e+04 | 3.765953e+02 |
| IFEP-Simple | -1.092814e+04 | 4.070844e+02 |
| EIFEP | -1.087647e+04 | 3.622485e+02 |
| IFEP-Simple-Local | -1.110368e+04 | 3.567361e+02 |

表 XXI
对比 7 种算法在 FUNCTION 9 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 8.298275e+01 | 2.195073e+01 |
| FEP | 3.670786e-02 | 7.560276e-03 |
| IFEP | 5.070210e-03 | 1.235013e-03 |
| IFEP-Single | 3.891059e-02 | 1.819514e-01 |
| IFEP-Simple | 7.157849e-02 | 2.522047e-01 |
| EIFEP | 2.756934e-01 | 5.898953e-01 |
| IFEP-Simple-Local | 3.016201e-01 | 5.918225e-01 |

表 XXII
对比 7 种算法在 FUNCTION 10 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 9.079882e+00 | 2.750298e+00 |
| FEP | 2.017164e+01 | 5.056674e-02 |
| IFEP | 4.401756e-03 | 4.660273e-04 |
| IFEP-Single | 4.568987e-03 | 3.375865e-04 |
| IFEP-Simple | 4.677204e-03 | 4.840279e-04 |
| EIFEP | 5.235960e-03 | 1.062181e-03 |
| IFEP-Simple-Local | 3.694432e-03 | 9.535198e-04 |

表 XXIII
对比 7 种算法在 FUNCTION 11 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | 1.619826e-01 | 3.104985e-01 |
| FEP | 1.880566e-02 | 1.911384e-02 |
| IFEP | 5.420949e-02 | 5.894522e-02 |
| IFEP-Single | 4.659441e-02 | 5.392762e-02 |
| IFEP-Simple | 3.826154e-02 | 4.871051e-02 |
| EIFEP | 5.300979e-02 | 5.457313e-02 |
| IFEP-Simple-Local | 9.845995e-03 | 1.265269e-02 |

表 XXIV
对比 7 种算法在 FUNCTION 13 上的性能

| Algorithm | Mean best | Std Dev |
|---|---|---|
| CEP | -6.616002e+00 | 2.795001e+00 |
| FEP | -5.055166e+00 | 3.713593e-05 |
| IFEP | -5.870670e+00 | 2.275363e+00 |
| IFEP-Single | -6.104369e+00 | 2.351020e+00 |
| IFEP-Simple | -6.490755e+00 | 2.538487e+00 |
| EIFEP | -4.891133e+00 | 1.776448e+00 |
| IFEP-Simple-Local | -7.283242e+00 | 2.795948e+00 |