

# 大连理工大学本科毕业设计（论文）

## 基于遗传编程的连续优化问题演化

Evolution of continuous optimization problem based on genetic programming

学 院（系）： 软件学院

专 业： 数字媒体技术

学 生 姓 名： 张金哲

学 号： 201692117

指 导 教 师： 任志磊

评 阅 教 师： 江贺

完 成 日 期： 2020 年 6 月 5 日

大连理工大学

Dalian University of Technology

## 原创性声明

本人郑重声明：本人所呈交的毕业设计（论文），是在指导老师的指导下独立进行研究所取得的成果。毕业设计（论文）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究成果做出重要贡献的个人和集体，均已在文中以明确方式标明。

本声明的法律责任由本人承担。

作者签名： 张金哲      日 期： 2020 年 5 月 30 日

## 关于使用授权的声明

本人在指导老师指导下所完成的毕业设计（论文）及相关的资料（包括图纸、试验记录、原始数据、实物照片、图片、录音带、设计手稿等），知识产权归属大连理工大学。本人完全了解大连理工大学有关保存、使用毕业设计（论文）的规定，本人授权大连理工大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业设计（论文）。如果发表相关成果，一定征得指导教师同意，且第一署名单位为大连理工大学。本人离校后使用毕业毕业设计（论文）或与该论文直接相关的学术论文或成果时，第一署名单位仍然为大连理工大学。

论文作者签名： 张金哲 日 期： 2020 年 5 月 30 日

指导老师签名： 何云 日 期： 2020 年 6 月 5 日

## 摘 要

本文提出了一种利用遗传编程生成目标函数的方法，函数的目标是使其在某个给定的已知进化算法上运行时性能明显优（或劣）于其在给定算法集上其他算法上的运行性能。本文使用了一组已建立的算法，在规定的函数维度以及规定的有限连续自变量空间下，对于每一个算法，进化都会产生一个唯一容易（或困难）的问题实例。每一个问题或实例只对一个算法有利（唯一简单或唯一困难）。针对现有的一系列算法性能度量方法，并综合各进化算法的特性，本文提出了新的更可靠的性能指标，并细化了运行性能指标的计算办法以使结果更加准确。本文将遗传编程的个体进行改进作为目标函数，依照算法集和最优化问题策略生成单一的性能指标，并以此为依据对目标函数进行演化。针对各参数下实验结果，本文对实验中各算法性能以及其各自擅长的问题环境范围进行分析和总结。通过遗传编程的手段，本文在保证生成目标函数足够有效的前提下，将生成的目标函数范围拓宽，增加了生成函数的形式的普适性。

**关键词：遗传编程；连续优化问题；进化算法；性能指标；目标函数**

## **Evolution of continuous optimization problem based on genetic programming**

### **Abstract**

In this paper, a method of generating objective function by genetic programming is proposed. The objective of the function is to make it run better (or worse) on a given evolutionary algorithm than on other algorithms on a given algorithm set. In this paper, we use a set of established algorithms. For each algorithm, evolution will produce a unique easy (or difficult) problem instance under the specified function dimension and the limited continuous independent variable space. Each problem or instance is only good for one algorithm (only simple or only difficult). Aiming at a series of existing algorithm performance measurement methods, and integrating the characteristics of each evolutionary algorithm, this paper proposes a new and more reliable performance index, and refines the calculation method of performance index to make the results more accurate. In this paper, the individual improvement of genetic programming is taken as the objective function, and a single performance index is generated according to the algorithm set and optimization strategy, and the objective function is evolved on this basis. According to the experimental results of each parameter, this paper analyzes and summarizes the performance of each algorithm in the experiment as well as the problem environment range that they are good at. By means of genetic programming, on the premise of ensuring that the generating objective function is effective enough, the scope of generating objective function is widened, and the universality of generating function form is increased.

**Key Words: Genetic Programming; Continuous Optimization Problem; Evolutionary Algorithms ; Performance Index; Objective Function**

## 目 录

摘    要 .....	I
Abstract .....	II
引    言 .....	1
1 进化算法性能度量 .....	4
1.1 本章简介 .....	4
1.2 新的性能指标 .....	4
1.3 算法单次对函数求解评定方法 .....	6
1.4 算法对函数求解综合评定方法 .....	8
1.5 UE/UD 策略下目标函数性能评定 .....	9
1.6 目标函数性能评定流程总结 .....	10
2 测试问题拟合方法 .....	13
2.1 本章简介 .....	13
2.2 终止符集 .....	13
2.3 函数集及函数复杂度控制 .....	13
2.4 函数再处理 .....	14
2.5 函数演化策略 .....	15
2.6 遗传编程演化流程总结 .....	15
3 实验结果一——目标函数生成 .....	19
3.1 本章简介 .....	19
3.2 指定算法集下二维函数的 UE 问题 .....	19
3.3 指定算法集下多维函数的 UE 问题 .....	20
3.4 指定算法集下二维函数的 UD 问题 .....	22
3.5 结论 .....	22
4 实验结果二——目标函数性能分析 .....	24
4.1 本章简介 .....	24
4.2 目标函数下各算法运行实验数据 .....	24
4.3 对 UE (DE) 问题剖析 .....	27
4.4 对 UE (GA) 问题剖析 .....	30
4.5 对 UE (PSO) 问题剖析 .....	32
4.6 结论 .....	34
结    论 .....	35

参 考 文 献.....	36
附录 A 实验参数符号表 .....	37
附录 B 遗传编程个体的演化策略 ( $m=40$ ) .....	38
附录 C 实验结果一部分原输出数据 ( $m=40, t=30$ ) .....	39
修改记录.....	45
致 谢.....	47

## 引 言

优化问题 (Optimization Problem, OP) 的一般提法是指在一定约束条件下求解一个目标函数的最大值 (或最小值) 问题, 这种问题通常是事先确定了自变量维数 (一般为多维) 以及函数的形式 (或者映射策略)。当变量为连续变量 (即在一定区间内可以任意取值) 时, 该类问题被称为连续优化问题 (Continuous Optimization Problem)。对于可以用公式表示的, 形式比较简单的连续优化问题, 通常采用求导或求偏导的形式进行计算, 由此得到的结果准确且计算时间可控。然而, 由于一些优化问题并没有最优的解, 或是要计算出最优的解要花费很大的计算量。面对这类问题, 一般的做法是通过一定的策略迭代缩小求得的问题解与真实最优解的差距, 通常称解决这类方法为优化算法。现阶段, 一些优化算法 (如 PSO, DE, GA 等) 经过不断改进已经得到了很好的效果。

然而, 在工程领域中产生的各种优化问题的形式和参数不同, 优化算法处理它们时的性能也不同。如 IEEE CEC 中各类竞赛特定基准的评判策略就曾经于 2017 年被质疑<sup>[1]</sup>, 由于环境条件对算法的执行效果的影响, 以唯一测试环境测试参赛算法的性能有失公正。由于每一种优化算法采取的优化策略不同, 优化算法面对各种环境会有不同的偏重。换言之, 每种算法都有适合自己解决的问题集, 其中一些算法在其对应问题集中性能会显著提高。因此, 生成测试问题, 使优化算法求解问题效果好或不好 (对应的测试问题分别被称为 UE/UD 问题), 有利于在实际生产问题依照真实生产需求正确选择优化算法, 以便缩短试错时间, 提高程序运行效率。

在本文中, 生成的测试问题的目标是: 使此优化问题在给定算法集中一个特定算法下运算时, 性能显著地优于或者劣于给定算法集中的其他算法, 并在函数拟合演化的过程中, 尽量把这个差异达到最大化。为了使研究更具有实际意义, 本文在生成测试问题之后应该考虑对测试问题的形式进行分析的可行性。

楼洋等人提出以测量新算法的性能为目的组成了一个基准测试套件<sup>[2,3]</sup>。套件的基准问题以高斯景观生成器的最大集 (MSG)<sup>[4]</sup>为基准问题空间通过调参获得, 使用差分算法 (DE)<sup>[5]</sup>作为演化算法。尽管仅使用高斯景观生成器解决可以模拟相当一部分问题, 而且通过构建峰或谷, 很容易保证生成问题具有最大(小)值, 从而保证问题的合理性。但其仍存在较强的局限性。首先, 假定最优化问题的定义域每一维无界限, 高斯景观生成器模拟的函数在任一维度自变量趋于无穷大时函数值都会趋近于 0, 这意味着高斯景观生成器永远不能处理自变量趋于无穷大时函数值非趋近于 0 时的最优问题, 如  $f(x, y) = x^2 + y^2$  求最小值。尽管一般情况下处理的目标函数是有界限的, 但也难免造成函数形式上的一些局限。其次, 高斯景观生成器不擅长构建高原和平原盆地等景观。



函数的局限性会影响求解最优化问题演化调参的效果,但更重要的是,高斯景观生成器生成的最优化问题很难具有代表性,这会大大降低后续通过这个问题结果对算法的适合场景推演的可能性。

此外,楼洋等人在论文中<sup>[2]</sup>采用了传统的以程序的运行时间作为评价算法性能指标的方法。对于一般问题,以运行时间作为性能指标最能反映单一程序的真实运行状况与实际性能。然而,对于连续最优化问题演化的策略来说,使用运行时间作为性能指标有几个比较明显的问题:

- (1) 算法的拟合时间与函数程序的内部结构复杂程度有关系。因此在选择函数程序时,这种关系会对选择过程产生一种误导。使用运行时间作为函数性能指标,不仅一定程度上背离了这个初衷,还有将函数演化得过于复杂的风险,导致整个程序的运行陷入停滞。
- (2) 计算算法求解函数程序的运行时间时,程序运行时间会受到运行环境波动性的影响。因为给定算法集中的算法求解本身就是一个概率事件,使用运行时间作为性能指标增加了算法评测时生成指标的不稳定性。
- (3) 使用运行时间作为算法拟合函数程序的性能指标时,算法集中不同算法对函数拟合作横向比较的意义不大。横向比较时,在同一时间点,不同算法拟合函数的当前状态的变量中仍有许多变量互相不一致,甚至不能由运行的时间推得遗传算法中种群进行到了哪一代。这使得对各个算法的适用环境剖析变得困难。

针对使用高斯景观生成器最大集生成的函数受局限的问题,本文提出了一种利用遗传编程生成目标函数的方法。遗传编程 (Genetic Programming, GP)<sup>[10]</sup>是进化算法 (Evolutionary Algorithm, EA) 的一个分支,致力于利用生物进化的思想生成计算机程序,在回归和分类等标准机器学习问题上取得了显著的成功。遗传编程与遗传算法区别在于:遗传算法用字符串来表达问题,而遗传编程采用层次化结构(通常为树状结构)表达问题。本文即利用了遗传编程的这个特点,利用遗传编程生成函数个体,每一个个体即对应了一个测试问题。然后计算该问题在 UE/UD 的策略下的适应度作为遗传编程算法的个体适应度,进而对连续优化问题进行演化。演化结束后,取适应度最优的个体作为最终的 UE/UD 问题。此外,本文对传统的遗传编程进行了一些改进以适应连续函数优化的问题。

针对使用运行时间作为性能指标的问题,在计算函数适应度时,本文创新性地提出以求解最优问题时的测试函数执行次数代替执行时间作为性能指标,即:如果算法集中的某个算法在某次拟合目标函数的过程中每个个体执行了 $n_A$ 次目标函数程序,本文称 $n_A$ 为测试函数执行次数,整个 UE/UD 的性能指标基于多次实验的多个 $n_A$ 生成。算法拟合

函数程序的运行次数与函数程序本身的执行状况相对独立，仅与函数输入与输出之间的映射关系有关，因此这种性能指标更加稳定，科学。在横向对各个算法的 UE/UD 问题进行比较时，以拟合次数为基准进行分析更加易于理解。如可以细化到函数的每一次运行时目标进化算法中种群的变化，拟合相同的次数目标函数结果的增益程度与速度，等等。

本文的主要贡献包括：

- (1) 提出一种新的性能指标计算方法。相比计算执行时间，计算执行次数有不受运行环境影响，易操作等优点。结合各目标算法特点，分析执行次数可以更深入地探知各算法的本质，从而探究算法各自擅长解决的问题的环境。
- (2) 提出一种新的基于遗传编程的目标函数生成方法。利用遗传编程本身的随机性，可以极大地增加了目标函数形式的灵活性，拓展了目标函数的生成空间，从而提升目标函数性能。
- (3) 本文对实验结果生成的目标函数进行了模拟分析。利用新方法生成实验结果，输出目标函数，函数维度为二维时输出目标函数图像。输出函数生成过程中的部分有研究价值的实验数据，并且输出在该目标函数下的各个函数的性能指数表。因为本文使用基于遗传编程的方法生成函数式，相较于高斯景观生成器最大集中的函数对人更易于理解，所以本文在生成数字形式的实验结果以外更多地生成了函数式形式的实验数据，以便后人对本文进行研讨以及实验的重现。针对实验结果，本文对各算法性能进行一定的具有实际价值的讨论。

在无特殊说明时，本文将默认以三个进化算法：差分进化算法 (Differential Evolution Algorithm, DE)<sup>[5]</sup>，遗传算法 (Genetic Algorithm, GA)<sup>[8]</sup>与粒子群优化算法 (Particle Swarm Optimization, PSO)<sup>[9]</sup>作为本文的目标算法集 $S_A$ ，即 $S_A = \{DE, GA, PSO\}$ 。为方便计算，函数默认都为有界函数，且上界与下界分别为 1 与-1（对于多维函数的每一维，其定义域都为 $[-1, 1]$ ）。由于优化问题对于最大值问题和最小值问题都是等价的（函数 $g(X) = -f(X)$ 求解最大值的问题实际上等价于对函数 $f(X)$ 求解最小值的问题），为了描述和程序运行的方便，本文默认将最优化问题按最小值问题处理。

论文的其余部分安排如下：第一章描述了指定 UE/UD 问题时对于输入函数的性能评测方法。第二章描述了基于遗传编程算法和第一章所生成函数性能指标的目标函数的演化过程。第三章，第四章为实验结果部分，第三章列出了各目标问题下拟合出的目标函数表达式及相应的性能指标，并针对结果目标函数的形式进行了一定的经验总结。第四章针对二维函数下的 UE 问题进行了更深入的实验数据研究，依此尝试解释第三章实验中二维函数下的 UE 问题目标函数表达式形式的合理性，将实验结论总结，泛化。

## 1 进化算法性能度量

### 1.1 本章简介

在生成目标函数之前，有必要统一各进化算法对于输入函数的性能评测方法，抽象来说，即生成一个评测系统，输入为进化算法和评测函数，输出为可以直观表现该算法对函数解决能力强弱的评测指标。直观理解下，最优化问题性能指标可以简单表述为“算法求得函数最优解所需要的时间”。然而在实际实现过程中，若直接使用该表述内容方法求得性能指标，一方面指标准确性会受到一定影响，另一方面算法实际运行过程中会出现一些问题，如求解的停滞等。

本章由小到大阐述进化算法性能度量的具体流程，在理论或实验的基础上证明流程中一些改进的合理性，并在最后一节对整章内容进行总结。

### 1.2 新的性能指标

由于以待测函数执行时间为性能指标存在一些局限，本文提出了一种新的性能指标：待测函数执行次数。简单地说，若记目标函数为 $f(x)$ ，则算法程序求最优解的过程中 $f(x)$ 的运行次数即作为性能指标。执行次数越少，性能指标值越小，效果越好。

接下来本文讨论以程序运行时间作为性能指标的合理性。

考虑进化算法 A 迭代指定次数  $n$  时所需要的时间 $t_A^n$ 。在特定的实验环境下，影响 $t_A^n$ 的主要因素包括算法本身运算机制，迭代次数  $n$ ，待测函数单次执行时间  $t_0$  以及参数设置（如种群中个体数  $p$ ）。对于大部分进化算法，理论上有：

$$t_A^n = n \times (p \times k_A \times t_0 + t_A) \quad (1.1)$$

其中  $k_A$ ,  $t_A$  是与进化算法种类相关的常量。 $k_A$  表示使用进化算法 A 时在进化算法两次迭代之间，对于算法中的每个种群个体所进行的函数计算数目。对于 DE, GA, PSO 三个算法， $k_{DE}=2$ ,  $k_{PSO}=k_{GA}=1$ 。 $t_A$  则表示进化算法通过已得出的待测函数运算结果进行种群调整迭代时的所需时间。

算法参数设置一定时，考虑算法与算法间的性能差异。在比较两个算法的性能差异时，常采用将两个拟合时间求比值的形式。假定进化算法  $A_1, A_2$  在优化待测函数 $f(x)$ 时分别执行了  $n_1, n_2$  次，那么两个算法性能的差异 $I$ 可以用

$$I = \frac{t_{A_1}^n}{t_{A_2}^n} = \frac{n_1 \times (p \times k_{A_1} \times t_0 + t_{A_1})}{n_2 \times (p \times k_{A_2} \times t_0 + t_{A_2})} \quad (1.2)$$

表示，将  $I$  对  $t_0$  进行求偏导得：

$$\frac{\partial I}{\partial x} = \frac{p \times (k_{A_1} \times t_{A_2} - k_{A_2} \times t_{A_1})}{n_2^2 \times (p \times k_{A_2} \times t_0 + t_{A_2})^2} \quad (1.3)$$

由 (1.3) 可以在理论上证明, 当  $\frac{k_{A1}}{t_{A1}}$  与  $\frac{k_{A2}}{t_{A2}}$  相差很大时,  $t_0$  对于  $I$  有不可忽视的影响。而对于已知进化算法来说, 不同算法中两者通常差别很大。假定算法池中仅有两个算法, 如果直接以  $I$  为性能指标输入到遗传编程算法进行迭代求解  $A_1$  的 UE, 当  $\frac{k_{A1}}{t_{A1}} \geq \frac{k_{A2}}{t_{A2}}$  时, 尽管在  $n_1, n_2$  不变的前提下,  $I$  也会随  $t_0$  增大而增大, 程序倾向于将运行时间短的函数个体作为目标函数。反之, 程序倾向于将运行时间长的函数个体作为目标函数。由此可见, 即使对于两个完全等价 (即当输入相同时输出一定相同) 的函数程序, 如果函数程序的运行复杂程度不同, 以时间为基础计算出的两个函数性能指标也不同。

举例说明, 考虑二维函数  $f(x_1, x_2) = x_1^2 + x_2^2$  以及二维函数  $g(x_1, x_2) = \sum_{i=1}^{10} (0.1 \times x_1^2 + 0.1 \times x_2^2)$  求最小值, 二者本身是等价的, 但是对于计算机程序而言, 两函数单次执行时间不同, 即  $t_g > t_f$ 。对于  $S_A = \{DE, GA\}$ , 有实验表明  $\frac{k_{DE}}{t_{DE}} \geq \frac{k_{GA}}{t_{GA}}$  且差异显著。假定两函数具有相同的  $n_1$  和  $n_2$ , 在求 DE 的 UE 问题时, 程序会误认为函数  $f$  比  $g$  更好, 这可能会导致目标函数过于简单而失去多样性。而在求 GA 的 UE 问题时, 程序会误认为函数  $g$  比  $f$  更好, 这可能会导致函数无限膨胀而影响程序运行效率。

两个函数在不同算法下的实际运行结果记录于表 1.1, 表 1.2。一方面, 运行结果中运行时间和次数的高度正比例关系一定程度上证明了公式 (1.1) 的准确性。另一方面, 实验数据中  $\frac{t_{DE}}{t_{GA}}$  从函数  $f$  到函数  $g$  的增长验证了: 在等价待测函数以不同形式呈现在程序中时, 以时间为指标的方案产生了错误判断。

另外, 连续优化问题演化的初衷是通过改变连续优化问题的输入输出关系 (对于二维函数来说是通过改变函数图像形状设置峰/谷的景观) 来达到性能差异的目的, 而不是通过改变函数程序内部的计算过程。使用运行时间作为函数性能指标造成的这种情况, 一定程度上与本文连续优化问题演化的目的不符。使用执行次数为性能指标的另两个优势是不受程序运行环境的影响以及易依据实验结果对算法进行深入剖析, 这两点在本文引言中有提及, 此处不再赘述。

因此, 本文提出采用进化算法中每个个体执行待测函数的次数  $n_A = n \times k_A$  作为性能指标。可以理解为: 算法  $A$  中每个个体在一次计算函数  $f(X)$  最小值的过程中进行了  $n_A$  次  $f(X)$  的运算, 则  $n_A$  即为  $(A, f(X))$  这个 “算法—函数对” 的性能指标。当然, 后续对这个性能指标会有一些处理, 后面小节会讲到。

由于对于等效函数而言, 相同的输入期望产生相同的输出, 对于同一算法  $A$ ,  $n$  的期望相同, 所以  $n_A$  的期望相同。即  $n_A$  仅受待测函数形状和进化算法性质的影响。

表 1.1 对于 $f(x_1, x_2)$ 各算法迭代指定次数所需时间（单位：秒）

结果迭代次数	DE	GA	PSO
1000 次迭代结果	0.2214	1.7316	0.1107
2000 次迭代结果	0.4354	3.4925	0.2251
3000 次迭代结果	0.6570	5.2994	0.3454
4000 次迭代结果	0.8816	7.0929	0.4602
线性拟合公式 (单位：秒/千次)	$y = 0.2202x - 0.0017$	$y = 1.7891x - 0.0687$	$y = 0.1169x - 0.0068$
R 平方值	0.9999	1	0.9999

表 1.2 对于 $g(x_1, x_2)$ 各算法迭代指定次数所需时间（单位：秒）

结果迭代次数	DE	GA	PSO
1000 次迭代结果	1.6629	2.3158	0.7872
2000 次迭代结果	3.4005	4.3988	1.6874
3000 次迭代结果	5.2561	6.6716	2.7127
4000 次迭代结果	7.0317	8.8928	3.2734
线性拟合公式 (单位：秒/千次)	$y = 1.7962x - 0.1527$	$y = 2.2004x + 0.0688$	$y = 0.8484x - 0.0057$
R 平方值	0.9999	0.9997	0.9873

实际上，当 $t_0$ 远大于 $t_A$ 时，以 $n_A$ 和 $t_A^n$ 作为计算依据是等效的，因为此时在求性能指标比值时产生的结果是相同的，这表明了新旧两个性能指标之间的强烈关联性，且在一定程度上可以证实使用函数的执行次数作为本文性能指标的合理性。

在之后本文的叙述中，“待测函数的执行次数”的说法默认为每个个体执行待测函数的次数。此外，为控制变量，本文提及的所有进化算法的个体数量 $p$ 都设置为一个固定值 $p_A$ 。

### 1.3 算法单次对函数求解评定方法

由于本文使用遗传编程作为生成函数手段，为了保证得到函数的随机性，生成的所有函数最优解都是事先未知的。若想得到算法单次求解函数最优解的计算次数，前提是事先综合各算法求得较精准的函数的最优解，并对是否计算到达最优解制定相应的准则。在不丧失一般性的前提下，本文将所有优化问题都视为最小化问题。

首先，本文先探究各遗传算法求解函数最优解时的性质。

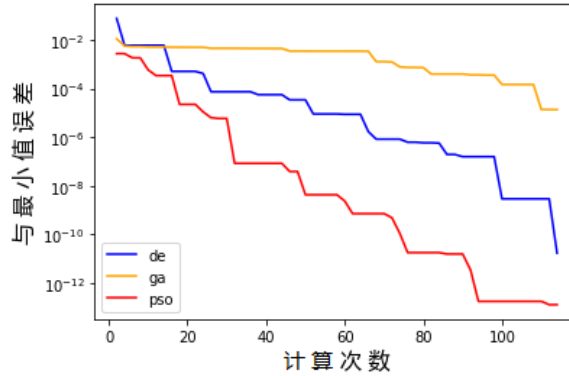


图 1.1 三个算法在求解 $f(x,y) = x^2 + y^2$ 最小值时  
误差与计算次数关系图

图 1.1 显示了三个算法在某次求解 $f(x,y) = x^2 + y^2 (x,y \in [-1,1])$ 最小值时的误差变化。为了使误差的变化更明显，折线图采用了对数刻度。易知 $f$ 最小值为 0，因此每次的计算结果即为所得误差。从三个图像对比分析，可大致得出结论：三个算法在计算 $f$ 时的性能指标排序为 $n_{PSO} < n_{DE} < n_{GA}$ ，即 PSO 性能最优，GA 性能最劣。

另外，对三个图像分别进行分析又可以得出两个经验：

- (1) 误差在计算过程中会出现多处中途“停滞”的情况（如 GA 算法在第 50 至 60 次计算误差没有变化）。这说明对于遗传算法，很多时候停滞并不代表计算的完成，尽管种群中长时间没有出现比之前最优解更优的个体，但整个种群仍然在演化并向更优的方向发展。
- (2) 误差的对数和计算次数成近似线性关系。以下讨论到达最优解的方法设计时，将以此作为参考。

针对遗传算法性质，本文设计了两个准确度参数 $\delta_C, \delta_B$ ，分别代表计算准确度和求解最优解的准确度。具体来说， $\delta_C$ 用于最优解求解及其他一些计算，当多个计算结果最大差值绝对值小于 $\delta_C$ 时，视作多个结果相等，本文称其为“计算相等”， $\delta_B$ 用于到达最优解评定，当计算结果与先前所求最优解的差小于 $\delta_B$ 时，视作计算结果得出了最优解，求解完成。理论上 $0 < \delta_C \ll \delta_B \ll 1$ 。

接下来本文讨论基于双准确度参数单次求解函数最小值评定方法，包括了最优解求解以及到达最优解评定方法。

首先讨论最优解的求解方法，由于 $S_A$ 本身就是求解最优解的算法集，本文使用 $S_A$ 中的所有算法对函数进行求解，求解停止条件是连续 $n_s$ 次待测函数运行前后结果“计算相等”，且每个算法求解多次（次数记为 $n_A$ ）以提高准确性。最后取这些最优解中最优的

一个结果作为最优解。这种做法会导致所求最小值和真实最小值的偏差，然而，即使用  $S_A$  以外算法求解，存在偏差也是必然的。当  $S_A$  中性能相差比较大时，在  $S_A$  解中取最优结果能保证所求最小值更接近真实值，而且依此求得的计算次数结果更能够反映出  $S_A$  算法间性能的差异。

最优解  $y_B$  求解之后，在对单次对函数求解的执行次数进行评定时，利用算法对被测函数进行计算，当计算结果不大于  $y_B + \delta_B$  时，返回求解成功的信息和待测函数的计算次数。当连续  $n_s$  次结果“计算相等”，但计算结果仍大于  $y_B + \delta_B$ ，说明该算法计算陷入停滞，返回求解失败的信息和陷入停滞以前的待测函数计算次数（即实际计算次数 -  $n_s$ ）

设计双准确度参数的意义是与单准确度参数方法（即  $\delta_B = 0$  或  $\delta_B = \delta_C$ ）相比，能够提高多次进行最小值评定时结果的稳定性，下面予以论述：

在评定前，函数在随机生成过程中最优解未知，需要事先求得，而所求结果  $y_B$  和真实值  $y_T$  之间的偏差是不可避免的。使用双准确度参数方法时，这个偏差  $\delta$  可以控制在  $\delta_C$  数量级范围内，记为  $k\delta_C$ （ $k$  为常数），即  $y_T = y_B - k\delta_C$ 。则在求各算法计算次数时的标准可记为  $y_C = y_B + \delta_B$ ，真实误差记为  $\delta_T = y_C - y_T = \delta_B + k\delta_C$ 。计算结果小于  $y_C$  即意味求解完成。利用上文中经验 (2)，假定已经进行了两次最小值独立评定，偏差分别记为  $k_1\delta_C$  与  $k_2\delta_C$ ，在求解计算次数时，计算数据相同的前提下，两次计算结果之间误差近似为  $k'(\log(\delta_B + k_1\delta_C) - \log(\delta_B + k_2\delta_C))$ （ $k'$  为常数）。对于双准确度参数方法，由于  $\delta_C \ll \delta_B$ ，可以认为这个误差等于 0，而对于单准确度参数方法，这项误差很可能存在。即双准确度参数方法几乎消除了评定最小值时的误差对最终结果的影响。

#### 1.4 算法对函数求解综合评定方法

在理想状态下，所有单次的求解信息都是成功的。然而在实际的计算过程中，存在某些算法在求解函数最优解时计算陷入停滞。导致这种情况可能的原因有两个：(1) 求解时算法中的种群全部陷入函数的局部最优解。(2) 求解时由于函数最优解附近的梯度过小而使最小值的计算过缓。相比 (1) 的情况，通常 (2) 中的算法经过有限次计算可以达到最优解。然而，在求解 UE/UD 问题时，生成函数的目标是体现已知各算法对其求解的差异，这导致在某些算法中函数出现 (2) 情况越明显，函数的性能越好，即导致了这些算法运行目标函数时时间的无限延长。

在上一节中，单次求解函数最优解评定方法返回了两个指标：布尔型变量表示单次求解是否成功，整型变量表示此次求解中计算待测函数的次数。求解函数最优解的综合评定方法，即通过多个单次求解函数最优解的评定结果，最终生成唯一的数值类型的性能指标。除了上述对陷入停滞的单次求解结果进行考虑之外，还应该考虑算法单次求解

函数最优解时计算次数的波动性。由于生成的初始种群以及进化路线的一些随机性，对于同样的算法——函数对，单次求解时函数执行次数有多有少，也可能有时陷入停滞，有时寻到最优解。综合考量时，性能指标主要与单次求解的平均计算次数以及陷入停滞的概率有关。

为描述方便，本文称求解成功的次数占求解总次数的百分比为“命中率”。一种综合评定的方法是求解固定的次数，之后计算单次求解的平均计算次数除以命中率，即计算出平均求解成功一次所需要的次数。这样做有两个问题：

- (1) 对于命中率比较高的函数，求解很少的次数就可以体现算法的性能。而对于命中率比较低的函数，求解同样的次数对应的求解成功次数很少，不足以反映算法性能。
- (2) 存在一些命中率几乎为 0 的算法——函数对，对于它们来说，如果求解永远不成功，求解次数则计算为正无穷大。如果因此输出的性能指标值为无穷大显然是不合理的。

针对问题 (1)， “求解固定的次数” 应考虑改进为“求解固定的成功次数”，但要考虑到命中率为 0 的情况，即不能因为求解一直不成功而无限地求解下去。针对问题 (2)， 应在命中率为零时做出相应调整措施，可以设置一个最小的命中率，使求得的命中率都高于这个最小命中率。

本文设置了三个参数，分别为求解成功次数上限 $n_V$ ，求解次数上限 $n_T$ ，以及最小命中率 $\alpha_M$ 。具体方案如下：

- (1) 每次综合评定开始时，通过 $S_A$ 计算函数的最优解，之后的单次求解时函数均以此作为最优解 $y_B$ 。
- (2) 之后对 $S_A$ 中每一个算法，反复地单次求解此函数，直至求解成功次数达到 $n_V$ 或求解总次数达到 $n_T$ 停止求解。
- (3) 计算出命中率 $\alpha$ 以及每次求解的平均计算次数 $\bar{n}$ ，将 $\bar{n} \times \frac{100\%}{\alpha + \alpha_M}$ 作为算法求解函数最优解的综合评定指标，记作 $fit_A$ （A 为对应的算法）。

本算法中，求解次数不小于 $n_T$ 和求解成功次数不小于 $n_V$ 两种情况，都视作了有足够的求解数据结果支撑综合评定结果。 $\alpha_M$ 为平滑因子，既保证 $\alpha$ 较大时综合性能指标与 $\alpha$ 成近似反比例关系，也保证了 $\alpha$ 较小时评定结果的稳定性。

## 1.5 UE/UD 策略下目标函数性能评定

记目标算法为 $A_T$ ，算法集 $B_A = S_A - \{A_T\}$ 为除目标算法外的其余算法集合。在 UE/UD 策略下，要使目标算法求解目标函数时性能明显优于/劣于其他算法，一种策略



是将 $A_T$ 性能与 $B_A$ 中所有算法性能平均值做比较, 另一种策略是将 $A_T$ 性能与 $B_A$ 中算法性能的最优/最劣值做比较。此外, HFEBG-H 算法<sup>[3]</sup>是一种综合考量 UE/UD 策略下求解指定函数时指定算法优劣性能的方法, 其在初期的拟合中具有一定优势。

HFEBG-H 算法输出三个参数, 而本文的算法目标是仅输出一个参数。因此, 本文对 HFEBG-H 算法进行简化, 仅取其第一部分, 即将 $A_T$ 性能与 $B_A$ 中最优/最劣值比较。具体实施方案如下:

- (1) 在 UE 策略下, 将 $A_T$ 的综合性能指标与 $B_A$ 算法中综合性能指标的最优值 (设为 $A_M$ ) 的比值作为目标函数评定结果。如 $A_T = DE$ ,  $S_A = \{DE, GA, PSO\}$ 时, 综合性能指标为 $\frac{fit_{DE}}{\min(fit_{GA}, fit_{PSO})}$ 。指标值越小, 目标函数越符合要求。
- (2) 在 UD 策略下, 将 $A_T$ 的综合性能指标与 $B_A$ 算法中综合性能指标的最劣值 (设为 $A_M$ ) 的比值作为目标函数评定结果。如 $A_T = DE$ ,  $S_A = \{DE, GA, PSO\}$ 时, 综合性能指标为 $\frac{fit_{DE}}{\max(fit_{GA}, fit_{PSO})}$ 。指标值越大, 目标函数越符合要求。

## 1.6 目标函数性能评定流程总结

本节结合前面所述内容, 对目标函数性能指标计算的一系列流程以伪代码的形式总结如下。代码中三个函数分别代表目标函数最小值求解办法, 单次对函数求解时的评定方法以及 UE/UD 策略下目标函数得分的计算方法。

---

### 算法 1 求解目标函数在指定 UE/UD 问题中的性能指标

---

输入:  $f_{aim}$  目标函数,  $S_A$  算法集,  $A_T$  目标算法,  $strategy$  问题策略 (UE 或 UD)  
输出: 性能指标

```

1: function CalcMin( $f_{aim}, S_A$ )
2:    $result \leftarrow +\infty$ 
3:   for  $A$  in  $S_A$  do
4:     for  $i = 1 \rightarrow 10$  do
5:        $temp\_min \leftarrow \text{CalcMinByAlgorithm}(A, f_{aim})$ 
6:       if  $result > temp\_min$  then
7:          $result \leftarrow temp\_min$ 
8:       end if
9:     end for
10:  end for
11:  return  $result$ 

```

```

12: end function
13:
14: function CalcMsgForOneTimeSolving ( $f_{aim}, A, \mathbf{S}_A$ )
15:    $real\_min \leftarrow \text{CalcMin}(f_{aim}, \mathbf{S}_A)$ 
16:    $\delta_B \leftarrow 10^{-10}$ 
17:    $n_S \leftarrow 400$ 
18:    $times \leftarrow 0$ 
19:    $population \leftarrow \text{InitPopulation}()$ 
20:    $res\_record \leftarrow \text{New Array}$ 
21:   while  $temp\_res > real\_min + \delta_B$  do
22:     if  $times > n_S$  and
23:        $res\_record[times] - res\_record[times - n_S] < \delta_B$  then
24:       return False,  $times - \delta_B$ 
25:     end if
26:      $times = times + 1$ 
27:      $population\_fitness \leftarrow \text{CalcOnceAimFunction}(population, f_{aim})$ 
28:      $res\_record[times] \leftarrow \min(population\_fitness)$ 
29:     EvolvePopulationByFitness( $population, population\_fitness$ )
30:   end while
31:   return True,  $times$ 
32: end function
33: function QuestionPerformanceIndex ( $f_{aim}, A_T, \mathbf{S}_A, strategy$ )
34:    $n_V \leftarrow 10$ 
35:    $n_T \leftarrow 200$ 
36:    $\alpha_M \leftarrow 0.005$ 
37:    $score_{A_T} \leftarrow 0$ 
38:    $score_{A_M} \leftarrow 0$ 
39:   for A in  $\mathbf{S}_A$ 
40:      $success\_solve\_times \leftarrow 0$ 
41:      $all\_solve\_times \leftarrow 0$ 
42:     while  $all\_solve\_times \leq n_V$  or  $success\_solve\_times \leq n_T$  do
43:        $is\_solve\_successful, times \leftarrow$ 
         CalcMsgForOneTimeSolving ( $f_{aim}, A, \mathbf{S}_A$ )

```

```

44:         if is_solve_successful then
45:             success_solve_times = right_times + 1
46:         end if
47:         all_solve_times = all_solve_times + 1
48:         total_calc_times = total_calc_times + times
49:         all_times = all_times + 1
50:     end while
51:     if  $A = A_T$  then
52:          $score_{A_T} \leftarrow total\_calc\_times / all\_solve\_times /$ 
           (success_solve_times/all_solve_times +  $\alpha_M$ )
53:     else if  $A_M = 0$  then
54:          $score_{A_M} \leftarrow total\_calc\_times / all\_solve\_times /$ 
           (success_solve_times/all_solve_times +  $\alpha_M$ )
55:     else if strategy = UE then
56:          $score_{A_M} \leftarrow \min (total\_calc\_times / all\_solve\_times /$ 
           (success_solve_times/all_solve_times +  $\alpha_M$ ),  $score_{A_M}$ )
57:     else
58:          $score_{A_M} \leftarrow \max (total\_calc\_times / all\_solve\_times /$ 
           (success_solve_times/all_solve_times +  $\alpha_M$ ),  $score_{A_M}$ )
59:     end if
60: end for
61: return  $score_{A_T} / score_{A_M}$ 
62: end function

```

---

本章介绍了一整套评测系统，输入为目标函数，初始算法集，目标算法以及对应的 UE/UD 策略。输出可以理解为这个目标函数在这套算法集与策略下的适应度，以浮点数形式表示。本章所介绍的系统为接下来的遗传编程推进问题打下了基础，在遗传编程的种群演化时，将使用此系统输出作为种群个体适应度指标。

## 2 测试问题拟合方法

### 2.1 本章简介

生成测试问题时，以遗传编程理论为基础，将函数问题以树状形式表示。生成新的初始问题的操作包括随机生成终止符和函数，并将它们组合为程序树，问题演化的操作包括交叉和变异，也包括生成新树（模拟种群的外来个体）。

本章介绍测试问题生成和演化方法，2.2, 2.3 介绍测试问题树的生成方法，2.4 将问题进行一些修正处理，2.5 介绍函数种群的演化方法，2.6 结合第一章，对 UE/UD 问题演化的整体流程进行总结。

### 2.2 终止符集

终止符集包括变量终止符和常量终止符。在函数的树节点为终止符节点的情况下，以固定的概率 $\alpha_v$ 和 $\alpha_c$ 分别生成变量和常量（ $\alpha_v + \alpha_c = 1$ ）。生成变量时，依据终止符的维度 $n$ 等概率地生成每个维度的变量 $x_i$ 。由于以正态分布生成常量过于复杂，本文未采用此方法生成常量。生成常量时，首先生成 (0,1) 范围内均匀的随机数 $m_r$ ，通过 sigmoid 函数的反函数将随机数映射到实数空间，生成对应常量 $m_c$ ，即：

$$m_c = -\log\left(\frac{1}{m_r} - 1\right)$$

### 2.3 函数集及函数复杂度控制

考虑到当节点函数自变量超出定义域的情况，本文函数集保证每个函数在整个实数集上都有定义。另外，为了保证生成连续的函数且函数值相对稳定，本文函数集中函数都为连续函数且自变量限制在一定数量级时函数值不会过大/过小。因此，函数集将对数运算进行了相应的改造，且没有使用除法运算。此外，考虑到生成的常量可能是负数，函数集并未添加减法函数和相反数函数，因为二者皆可以找到用乘法，加法生成的等价函数。

本文设置了以下函数集 $A_f$ （设 $X_i$ 为自变量）：

$$A_f = \{X_1 + X_2, X_1 \times X_2, \sin X_1, \cos X_1, \log(\text{abs}(X_1) + 1)\}$$

下文分别称 $A_f$ 中的五个运算为加法运算，乘法运算，正弦运算，余弦运算和对数运算。

每个节点分别以概率 $\alpha_f$ 和 $\alpha_e$ 生成函数和终止符（ $\alpha_f + \alpha_e = 1$ ）。生成函数时，等概率地抽取 $A_f$ 中的函数作为该节点的运算，并依据函数自变量的维度生成相应数量的子树作为该节点的子节点。

为使生成的函数树控制在一定的规模，本文采用以函数树深度抑制函数生成的方法：当节点为根节点时（此时节点深度为 0）， $\alpha_f = 1$ 。当节点深度为 $d$ 时， $\alpha_f = \max(1 - k \times d, 0)$ ， $k$ 为固定的抑制因子。即随着节点深度增加，节点生成终止符的概率增大，生成函数的概率减小，从而控制了函数树的规模。

## 2.4 函数再处理

通常情况下，函数树生成后其对应的函数可以用来进行最优值的求解。然而，这样生成的函数有一个问题：由于遗传编程的随机性也伴随着不稳定性，函数的最小值很可能落在函数定义域的边界，即函数图像可能会“过于倾斜”。由经验得知，在一些算法（如 PSO）中，对于这样的函数拟合的速度过快。在评定时，这些算法会在这些“过于倾斜”的函数得到异常好的性能指标。尽管遗传编程算法产生此类函数本身没有什么问题，但生成的倾斜函数过多将会使得那些在倾斜函数上表现不好的算法找到性能指标好的目标函数的概率非常小，时间过长。

因此，本文对已经生成的个体函数做了简单的处理。处理后的函数最小值更可能落在定义域的内部，但也存在可能性最小值仍落在函数的边界，以此在保证生成函数多样性的前提下解决这个问题。

因为函数处理的目标是在尽量不失函数随机性的前提下让函数的最小值点落到定义域边界的概率减小，对于自变量 $X$ 的一维 $X_i$ 来说，即应避免函数在 $X_i$ 等于下界时的函数值普遍远大于或远小于函数在 $X_i$ 等于上界时的函数值。本文的策略是取自变量其他维 $X_j$ 的值为上下界的平均数（即取中点）时，保证 $X_i$ 等于上界，下界时函数值相同。采用的方法是对于 $n$ 维的问题，添加 $n$ 个加项。对于第 $i$ 个加项，形式为 $+kX_i$ ，并计算参数 $k$ 满足上述策略。如，当生成个体对应的原始函数为 $f(X) = X_1^3 + \sin X_2$ （ $X_1, X_2 \in [-1, 1]$ ）时，处理后的函数为 $f(X) = X_1^3 + \sin X_2 - X_1 - (\sin 1) \times X_2$ （ $X_1, X_2 \in [-1, 1]$ ），从而保证了 $f\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = f\left(\begin{bmatrix} -1 \\ 0 \end{bmatrix}\right)$ ， $f\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) = f\left(\begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$

为方便说明，本文分别称这种处理前后的函数式为“原始式”与“处理式”。需要注意的是，由于原始式可以转换为处理式且表达式文本相对简单，在本文实验数据中（第三章及附录 C）函数式显示都为原始式，但是实验实际进行计算时使用处理式进行计算。

## 2.5 函数演化策略

本文函数演化策略主要基于遗传编程的原理。在计算各个个体性能指标之后每代依据指标的大小选择个体。在 UD 问题中，性能指标越大的个体性能越优，在 UE 问题中则是性能指标越小的个体性能越优。

演化方法使用到了交叉，变异生成新的个体。在本文中，交叉操作指的是一棵树（主树）的若干个随机节点分别替换为另一棵树（辅树）的若干个随机子树，变异操作指的是一棵树的若干个随机节点分别替换为新生成的随机树。为了方便描述，本节将交叉或变异的节点称作“目标节点”。

目标节点在选择时，流程与 2.3 节的方法有相似之处。设置  $k_v$ ， $k_i$  分别为函数变异与交叉的概率因子，对于初始树的根节点，将其选为目标节点的概率为 0，当节点深度为  $d$  时，对于变异与交叉操作，此节点选为目标节点的概率分别为  $k_v \times d$ ， $k_i \times d$ （当进行交叉操作时，主树和辅树交叉节点的选择都使用上述策略且以  $k_i$  为概率因子）。

进行交叉操作时，由于没有新生成的树，只需要将原子树按照原形式复制到新节点上。进行变异操作时，新子树的生成方法需要依照 2.3 节方法利用抑制因子  $k$  计算  $\alpha_f$  和  $\alpha_e$ ，因此新子树的生成与母树目标节点的深度有关。

对于种群的每一次演化，大体的步骤如下：

- (1) 对各个个体依性能指标进行排序。
- (2) 在下一代个体中，分别引入：
  - ① 上一代中排名前 12.5% 的优良个体
  - ② 上一代中排名前 12.5% 的个体随机节点互相交叉后生成的个体，排名越靠前，引入的子交叉个体越多。
  - ③ 上一代中排名前 10% 的个体随机节点变异后生成的个体，排名越靠前，引入的子变异个体越多。
  - ④ 上一代中排名前 75% 的个体随机节点互相交叉后生成的个体（数量较少）
  - ⑤ 新生成个体（模拟外来物种，数量较少）

具体的遗传编程个体演化策略参见本文附录 B。

## 2.6 遗传编程演化流程总结

本节总结遗传编程演化的整体流程，整理内容仍然以伪代码的形式展现。算法中函数分别代表了生成新个体方法，个体对应的目标函数方法，寻找随机子树方法，变异，交叉方法以及种群最终的演化策略。

算法的最终返回结果即为经演化  $m$  代以后种群内最优个体的对应目标函数。

---

**算法 2** 基于遗传编程的目标问题演化

---

输入:  $S_A$  算法集,  $A_T$  目标算法,  $strategy$  问题策略 (UE 或 UD),  $A_f$  运算符集,  
 $n$  问题维数

输出:  $f_{aim}$  目标函数

```

1: function InitTree( $root = \text{False}, depth = 0$ )
2:    $k \leftarrow 0.1$ 
3:    $\alpha_v \leftarrow 0.8$ 
4:    $func\_property \leftarrow 1 - k \times depth$ 
5:    $node \leftarrow \text{New Node}$ 
6:   if  $\text{rand}() < func\_property$  then
7:      $node.kind \leftarrow "function"$ 
8:      $node.func \leftarrow \text{RandomChoiceFrom}(A_f)$ 
9:     for  $i = 1$  to  $\text{ArgumentNumber}(node.func)$ 
10:       $node.children[i] \leftarrow \text{InitTree}(node, depth + 1)$ 
11:    end for
12:     $node.expression \leftarrow \text{CombineFuncExpression}(\mathbf{}$ 
13:       $node.func, node.children)$ 
14:  else if  $\text{rand}() < \alpha_v$  then
15:     $node.kind \leftarrow "variable"$ 
16:     $node.expression \leftarrow \text{CombineVarExpression}(\text{RandomInt}(n))$ 
17:  else
18:     $node.kind \leftarrow "const"$ 
19:     $node.expression \leftarrow \text{CombineConstExpression}(\text{rand}())$ 
20:  end if
21:  return  $node$ 
22: end function
23:
24: function NodeCalc( $node, args$ )
25:   return  $\text{Calc}(\text{GetFuncFromExpression}(node.expression), args)$ 
26: end function
27:
28: function SubTree ( $node, probability\_factor, depth = 0$ )
29:   if  $node.kind \leftarrow "variable"$  or  $node.kind \leftarrow "const"$  then

```

```

30:         return node
31:     end if
32:     if rand() < probability_factor × depth then
33:         return node
34:     end if
35:     return depth, SubTree (RandomChoiceFrom(
                                   (node.children), probability_factor, depth + 1)

36: end function
37:
38: function Variation(treeroot)
39:      $k_v \leftarrow 0.1$ 
40:     depth, subnode ← SubTree(treeroot,  $k_v$ )
41:     InitTree(subnode, depth)
42: end function
43:
44: function Intersect(tree1_root, tree2_root)
45:      $k_i \leftarrow 0.1$ 
46:     depth1, subnode1 ← SubTree(tree1_root,  $k_i$ )
47:     depth2, subnode2 ← SubTree(tree2_root,  $k_i$ )
48:     NodeReplace(subnode1, subnode2)
49: end function
50:
51: function GeneticProgrammingEvolution()
52:      $m \leftarrow 40$ 
53:     for  $i = 1$  to  $m$ 
54:         population[ $i$ ] ← InitTree()
55:         population[ $i$ ].score ← QuestionPerformanceIndex(
                                   population[ $i$ ].NodeCalc,  $A_T$ ,  $S_A$ , strategy)
56:     end for
57:     SortByScore(population)
58:      $t \leftarrow 30$ 
59:     for  $i = 1$  to  $t$ 
60:         EvolveByIntersectAndVariationFuncAndScore(population)
61:         RefreshScore(population)

```



```
62:         SortByScore(population)
63:     end for
64:      $f_{aim} \leftarrow population[0].NodeCalc$ 
65:     return  $f_{aim}$ 
66: end function
```

---

本章介绍了基于遗传编程的测试问题生成方法，并结合第一章内容形成了整套的目标函数演化方法。通过遗传编程演化策略，既保证了测试问题多样性，又保证了现有种群的适应度不断提高。因此，最终生成的目标函数在形式上受人为约束小，在性能上也达到了实验预期。

### 3 实验结果——目标函数生成

#### 3.1 本章简介

在本章中，我们使用基于遗传编程的组合算法对连续优化函数进行演化，求解确定维度条件下的指定算法的 UE/UD 问题目标函数。

本章分别以不同策略对不同维度函数进行演化。其余参数设置为相同的值，其中包括：

- (1) 进化算法种群中个体数量 $p_A$
- (2) 双准确度参数 $\delta_C, \delta_B$ ，连续计算相等次数容忍度 $n_s$
- (3) 求解成功次数上限 $n_V$ ，求解次数上限 $n_T$ ，最小命中率 $\alpha_M$
- (4) 遗传编程树节点生成函数概率的抑制因子 $k$ ，生成变量和常量的概率 $\alpha_v, \alpha_c$ ，变异与交叉的概率因子 $k_v, k_i$
- (5) 遗传编程种群数量 $m$ 以及演化次数 $t$ 。

参数具体值参见附录 A。

本文的更多实验数据参见附录 C。需要注意的是，在无特殊说明的情况下，本章及附录 C 中实验数据函数式均采用“原始式”（见 2.4 节）方法表示。

本节内容安排如下：3.2 节讨论二维函数在算法集 $S_A$ 下 UE 问题的实验结果。3.3 节提升了目标函数问题的维度，讨论 $n = 10$ 时目标函数的 UE 问题实验结果。3.4 节讨论二维函数在 UD 问题中的实验结果。3.5 节结合实验的最终目的进行了一些经验总结。

#### 3.2 指定算法集下二维函数的 UE 问题

表 3.1 列出了使用基于遗传编程的组合算法生成的二维 UE 问题目标函数的原始式。表中三个目标函数分别对应了三个 UE 问题，即：在 $X_1, X_2 \in [-1, 1]$ 的条件下求解目标函数最小值的问题。对于初始算法集 $S_A = \{DE, GA, PSO\}$ ，下面每一个问题都是通过选择 $S_A$ 中三个 EA 中的一个作为目标 $A_T$ 而产生的。从表中数据来看，对于每个问题实例，目标函数性能指标差异明显，这意味着对于 UE 问题来说，目标函数的形态会极大地影响算法的性能。

由于本文的函数性能指标仅仅考虑函数的运行次数，所以本文的实验结果没有考虑运行环境，计算复杂程度等条件的影响。如果使用目标函数最优问题的运行时间作为性能指标，性能指标的值将会进一步减小。然而，目标函数的运行次数已经能够充分并科学地显示出函数形态对函数性能的影响性。

表 3.1 在 $S_A$ 算法集条件下二维函数的 UE 问题

问题类型	目标函数处理式	目标函数性能指标
UE (DE)	$y = \log(\text{abs}(\sin(\log(\text{abs}(X_1 + X_2) + 1))) + 1) + \sin(\sin(X_1))$	$2.2525 \times 10^{-2}$
UE (GA)	$y = \sin(\log(\text{abs}(X_2 + \sin X_1) + 1))$	$6.3932 \times 10^{-1}$
UE (PSO)	$y = \sin((\cos((0.2317)) + \cos((X_1 + \cos(\sin((\sin(X_2) \times X_2))))))$	$4.5816 \times 10^{-3}$

将目标函数的性能指标结合在本文 1.3 节中三个算法在求解  $f(X) = X_1^2 + X_2^2$  ( $X_1, X_2 \in [-1, 1]$ ) 最小值时的误差来分析, 并假定 1.3 节中的实验结果为三个算法求解二维函数的普遍性能指标。PSO 算法对函数形态最敏感: UE (PSO) 目标函数相比于  $f(X) = X_1^2 + X_2^2$  性能指标下降了 98% 以上, 改变最明显。GA 算法性能指标普遍最劣, 在二维问题的 UE (GA) 性能指标的改善幅度也最小, 导致对于 UE (GA) 目标函数  $A_T$  相比  $A_M$  的性能差异相对不明显。

分析各个算法的 UE 问题目标函数原始式表达式形式, 结合附录 C, 可以得出经验: DE, GA 算法对于加法, 对数运算比较多的函数运行的性能较优, PSO 算法对于乘法, 余弦运算比较多的函数运行的性能较优。另外, PSO 算法目标函数表达式有时会表现为若干个子函数连乘的形式。更具体各个算法的目标函数形式分析将在第四章描述。

### 3.3 指定算法集下多维函数的 UE 问题

表 3.2 列出了维数为 10 时使用基于遗传编程的组合算法生成的目标函数原始式。函数的生成方法和其他的参数与 3.1 中实验相同。对于维数增加的情况, 可以从表中看出自变量的范围由  $X_1 \sim X_2$  拓展到  $X_1 \sim X_{10}$ 。然而, 并不是每一维的自变量  $X_1 \sim X_{10}$  都涵盖在 UE/UD 问题的函数表达式中。这是因为本文 2.2 节在生成终止符策略中, 生成变量本身是一个概率事件, 由于函数树的规模有限, 不能保证节点一定涵盖所有维度的变量。

首先观察目标函数的性能指标。对于 GA 算法和 PSO 算法, 求解 UE 问题时随着维度的升高, 目标函数的性能指标都得到了明显的改善。PSO 的性能指标下降为处理二维函数问题时的 22.37%, GA 的性能指标下降为处理二维函数问题时的 28.96%。对于 UE (GA) 来说,  $A_T$  相比  $A_M$  的性能差异在维度为 10 的问题中相比二维问题的效果要好很多。

然而, 对于 DE 算法来说, 目标函数的性能指标没有下降, 反而比二维问题中的性能指标上升了一倍左右。这里解释一下为什么理论上维数上升性能应该更好: 对于维数  $n = 2$  时的 UE (DE) 目标函数, 也可以将其看作是一个  $n = 10$  的函数, 只是函数值不随  $X_3 \sim X_{10}$  的改变而改变。若将  $n = 2$  时的 UE (DE) 目标函数使用  $n = 10$  时的 DE, GA, PSO 算法进行求解, 其结果等效于使用  $n = 2$  时的 DE, GA, PSO 算法进行求解。也就是说,

如果把函数 $y = \log(\text{abs}(\sin(\log(\text{abs}(X_1 + X_2) + 1))) + 1) + \sin(\sin(X_1))$ 认作一个 $n = 10$ 的问题的话，性能指标比本次实验中利用 $n = 10$ 本身求解问题得到的函数式还要好。

表 3.2 在 $S_A$ 算法集条件下维数为 10 的函数 UE 问题

问题类型	目标函数表达式（遗传编程种群最优解）	目标函数性能指标
UE (DE)	$y = \cos(\cos(X_7)) + \log(\text{abs}(\log(\text{abs}(\log(\text{abs}(X_7 + X_8) + 1)) + 1) \times \sin(\log(\text{abs}(0.4555) + 1))) - 0.5353$	$4.5326 \times 10^{-2}$
UE (GA)	$y = \log(\text{abs}((((\sin(X_8) \times (\sin(X_1) + (X_2 + \cos(X_{10})))) \times (\log(\text{abs}((-0.8565)) + 1) \times X_9)) + (X_1 + (((X_7 \times X_1) + ((X_{10} + X_1) + X_3)) \times X_4 \times ((X_6 + \cos(X_9)) + (X_1 + X_6)))))) + 1)$	$1.8513 \times 10^{-1}$
UE (PSO)	$y = (X_6 \times (X_6 + X_4)) \times (X_{10} + (\cos(X_3) \times \cos(0.2465)))$	$1.0248 \times 10^{-3}$

这个问题的解释可以从目标函数表达式的形式中找到。即使问题维数为 10, UE (DE) 问题的目标函数仍是一个实际意义上的二维函数。因此，本文可以得到经验：相比其他进化算法，DE 算法更适合解决维数比较小的一类问题，即随着函数的维度的增大，UE (DE) 问题的性能指标很可能降低。

对于遗传编程来说，虽然生成函数的形态极具多样性，但是相应地在生成函数数量一定时函数与函数间的形态变化过大，即生成的函数相对于整个函数空间比较“稀疏”。因此，对目标函数的求解过程应该看作一个概率事件。即使维数为 10 条件下能比维数为 2 条件下生成更具多样性的问题，但是在相同的性能条件下，维数为 10 条件下生成的低维目标函数的概率要远远小于 $n = 2$ 时的情况。假定上面得出的经验是正确的，那么对于 UE (DE) 问题，维数为 10 条件下生成的优质的目标函数个体数目自然要少于二维问题下优质目标函数个体数目，进而导致多维问题寻找优质目标函数变得困难。

相比之下，GA 算法在多维问题上的表现比较突出，对于 UE (GA) 问题来说，生成的目标函数是一个实际意义上的八维函数。当问题的维数增大的时候，UE (GA) 的目标函数在形式上明显地变得更复杂（体现在函数式上长度更长），且实际变量数量明显增多，性能指标明显改善。

横向比较下，目标函数表达式的形式的其他方面的比较与二维函数相比仍有相似之处，UE (PSO) 的目标函数仍然是余弦和乘法的运算比较多，UE (DE) 和 UE (GA) 的目标函数仍然是对数和加法的运算比较多。DE, GA 算法对于加法，对数运算比较多的函数运行的性能较优，PSO 算法对于乘法，余弦运算比较多。总的来说，维数为 10 的函数 UE 问题性能指标整体有所改善。

### 3.4 指定算法集下二维函数的 UD 问题

表 3.3 列出了对于 UD 问题来说三个算法生成的目标函数的原始式。UD 问题的目标函数在不同算法的性能指标，计算方法为目标算法的综合性能指标与算法集中其余两个算法的最差（值最大）综合性能指标的比值。由于这个值越大越好（目标算法性能越差越好），目标函数的性能指标通常大于 1。

表 3.3 在  $S_A$  算法集条件下二维函数 UD 问题

问题类型	目标函数表达式（遗传编程种群最优解）	目标函数性能指标
UD (DE)	$y = \log(\text{abs}(((\cos((X_2 + (\sin(((X_2 \times X_2) + 0.4652)) \times \sin(X_2)))))) + (\log(\text{abs}(X_2) + 1) \times (\log(\text{abs}((\cos(X_2) \times X_1)) + 1) \times \sin(2.1511)))) + \sin(-2.8648))) + 1)$	$6.0460 \times 10^1$
UD (GA)	$y = ((\log(\text{abs}((X_2 + (X_1 \times X_1))) + 1) + \log(\text{abs}(\cos(X_1)) + 1)) + \log(\text{abs}((\cos((\cos((2.9009)) + \cos(X_1))) \times \sin(X_2))) + 1))$	$2.1012 \times 10^2$
UD (PSO)	$y = (\sin(X_2) + \sin(\log(\text{abs}(((\sin((X_1 + X_1)) + X_2) \times (1.8838))) + 1))))$	$6.4143 \times 10^0$

分析各个算法对应的性能指标值，可以看出各算法在其各自目标函数上的求解效果普遍都很明显。性能指标最大值为 UD (GA)，最小值为 UD (PSO)，依据前文的经验，GA 算法性能最劣，PSO 算法性能最优，因此对于 UD 问题目标函数性能指标的值大小关系也符合预期。相较于二维函数 UE 问题的实验结果，UD (GA) 与 UE (PSO) 的性能指标的优劣程度相似，但 UD (PSO) 相较 UE (GA) 的性能指标明显更优一些。

对目标函数原始式进行分析：对于 UD 问题来说，UD (DE) 与 UD (GA) 所对应的函数表达式余弦运算比较多，UD (DE) 对应的乘法运算比较多。而 UD (PSO) 和 UD (GA) 对应的加法的运算比较多。这些经验也与本文进行 UE 问题求解的实验结果相呼应。

### 3.5 结论

对连续函数的优化问题进行演化，最终的目标是在实际生产问题中，依照生产需求正确选择优化算法以提高程序运行效率。因此，对实验结果的总结工作必不可少。本章前面已经对实验结果进行了相当一部分总结。本节结论部分依据前面的经验，简要地总结今后面对不同环境下连续优化问题时选择  $S_A$  中算法的注意事项。

对于乘法，余弦运算比较多的函数式，或者与这种函数式图像比较相近的函数映射关系，建议使用  $S_A$  算法中的 PSO 算法。对于加法，对数运算比较多的函数式，或者是维数比较高的函数问题，建议使用  $S_A$  算法中的 GA 算法。对于函数维数比较少的问题，可

以使用 $S_A$ 算法中的 DE 算法。当问题函数式没有显著特征时，通常情况下考虑使用 PSO 算法进行连续函数优化。

## 4 实验结果二——目标函数性能分析

### 4.1 本章简介

如果想对算法集中的所有算法适合的目标函数特征进行原理的总结，只通过连续优化函数的演化策略生成目标函数是不够的。尽管在上一章中，我们通过目标函数的形式和最终的性能指标能够提出合理的猜想，但是由于是从实验结果得到的经验，所以难免缺乏科学性，在准确性上也会较差。

本文利用基于函数执行次数的原理进行连续优化函数演化，对于研究者来说其最大的好处就是可以更直观地进行各算法间的横向比较。例如，在 1.3 节中本文以函数  $f(x, y) = x^2 + y^2$  为例，使用三个算法进行单次函数作为对三个算法的一个直观的理解，且图中清晰地反映了三个算法的性能差异。

本章针对在  $S_A$  下二维函数的 UE 问题进行了细化的研究，研究顺序由表及里。其中包括：

- (1) 目标函数下各算法运行的性能指标
- (2) 目标函数下各算法运行每次求解平均计算次数，求解成功率
- (3) 各目标函数的形态及各算法运行各目标函数时误差变化

### 4.2 目标函数下各算法运行实验数据

表 4.1 列出了在  $S_A$  算法集下二维函数的 UE 问题处理式，分别对应了表 3.1 中 UE 问题的原始式。由于第四章整章涉及进化算法对函数的计算，而实际参与运算的是各问题目标函数的处理式，故本章列出此表。处理式中粗体部分为附加项。因为本文遗传编程所用函数集  $A_f$  中的函数大都具有一定的对称性特点，所以会出现一些附加项为 0 的情况。

表 4.1  $S_A$  算法集下二维函数的 UE 问题处理式

问题类型	$X_1$ 附加项系数	$X_2$ 附加项系数	目标函数处理式 (遗传编程种群最优解)
UE (DE)	-0.7456	0	$y = \log(\text{abs}(\sin(\log(\text{abs}(X_1 + X_2) + 1)))) + 1) + \sin(\sin(X_1)) - \mathbf{0.7456} \times X_1$
UE (GA)	0	0	$y = \sin(\log(\text{abs}(X_2 + \sin X_1) + 1))$
UE (PSO)	0.1957	0	$y = \sin((\cos((0.23171727502870929)) + \cos((X_1 + \cos(\sin((\sin(X_2) \times X_2))))))) + \mathbf{0.1957} \times X_1$

表 4.2 列出了三个 UE 问题分别在各个算法上的综合性能指标以及求得的性能指标。由 1.5 节描述可知, 对于  $S_A$  算法集 UE 问题的性能指标计算方法为  $\frac{A_T}{A_M}$ 。由于表 4.2 与表 3.1 中实验分别对应了相同目标函数的两次独立评测程序运行, 所以目标函数性能指标会有一定的偏差。

由表 4.2 可以看出, 对于 UE (DE) 与 UE (GA) 的问题而言, 目标函数达成 UE 问题的主要手段为增加  $B_A$  中两个算法的综合评定指标。而对于 UE (PSO) 问题而言, 目标函数达成 UE 问题的主要手段为减少  $A_T$  算法的综合评定指标。

仅凭借本表所得结果, 可能会得出一些错误的经验, 因为影响这些评定指标的因素并不唯一。因此, 本表应结合计算这些指标的数据来分析。

表 4.2 在  $S_A$  算法集条件下二维函数的 UE 问题  
对应的各算法综合性能指标

问题类型	各算法下的综合评定指标			目标函数性能指标
	$fit_{DE}$	$fit_{GA}$	$fit_{PSO}$	
UE (DE)	1126.8026	48888	187729	$2.3049 \times 10^{-2}$
UE (GA)	6982.4975	247.7612	405.7711	$6.1059 \times 10^{-1}$
UE (PSO)	313.4328	384.2786	1.1940	$3.8095 \times 10^{-3}$

表 4.3 列出了三个 UE 问题中各个算法运行时的实际信息, 包括每次求解时待测函数运行次数的均值, 总求解次数, 求解成功次数与求解成功率。由本文 1.3 节内容, 结合附录 A 参数表可知, 本文计算指标的策略为总求解次数达到 200 或求解成功次数达到 10。由于设置了最小命中率  $\alpha_M = 0.005$  作为平滑因子, 即使算法求解目标函数成功率为 0% 也可以依据求解失败的平均次数得到一个性能指标。

由表中可以看出, 求解成功率对算法求解函数时的性能指标起到很重要的影响。对于某些 UE 问题来说, 求解成功率的影响力甚至超过了待测函数的平均运行次数的影响力。

对于 UE (DE) 问题来说, 目标函数使 UE 问题性能指标提升的主要手段是增大 DE 算法的求解成功率, 而减少 GA, PSO 算法的求解成功率, 即, UE (DE) 问题生成了一个用 DE 算法解决可能性非常大, 而用 GA, PSO 算法解决可能性都非常小目标函数。UE (DE) 问题在 GA 和 PSO 算法下的求解成功率降至 0%, 因而极大地增大了  $fit_{GA}$  与  $fit_{PSO}$  的值。而由于求解成功率为 0%, 将  $fit_{GA}$  与  $fit_{PSO}$  进行比较, 很难说明两个算法在



执行 UE (DE) 目标函数的性能差异。这仅能说明执行 UE (DE) 目标函数时,相较于 PSO 算法,GA 算法能够更快地进入计算停滞状态,从而减少了每次计算时试错的次数。而对于 UE (GA) 与 UE (PSO) 问题来说,目标函数提升 UE 问题性能指标的主要手段是提升待测函数平均运行次数的差异。对于 UE (GA) 问题,指标计算的决定性因素是 GA 算法与 PSO 算法运行此问题之间的性能差异。然而,对于 GA 算法求解成功率较高的目标问题,对于 PSO 算法求解成功率也比较高。因此,UE (GA) 问题的性能提升受到限制。而对于 UE (PSO) 问题来说,由于 PSO 算法对个别问题的性能极强,本文 UE 的性能指标又是以比值的形式计算的,因此解决 UE (PSO) 问题的关键在于找到在 PSO 算法下运行效果极好的个体,在于减少 PSO 算法中待测函数平均运行次数。

表 4.3 在  $S_A$  算法集条件下二维函数的 UE 问题  
对应的各算法求解函数信息汇总

目标问题类型	指标类型	使用的进化算法		
		DE	GA	PSO
UE (DE)	待测函数平均运行次数	1030	244.44	938.645
	总求解次数	11	200	200
	求解成功次数	10	0	0
	求解成功率	90.909%	0%	0%
UE (GA)	待测函数平均运行次数	1044.2941	124.5	203.9
	总求解次数	34	10	10
	求解成功次数	10	10	10
	求解成功率	29.412%	100%	100%
UE (PSO)	待测函数平均运行次数	315	386.2	1.2
	总求解次数	10	10	10
	求解成功次数	10	10	10
	求解成功率	100%	100%	100%

结合表 4.2 与表 4.3, 针对三个算法可以得出以下结论:

- (1) 对于 PSO 算法而言, 其在求解 UE (PSO) 目标函数平均仅仅计算了不到两次, 这说明 PSO 算法对于个别的问题具有极强的求解能力。而且, PSO 算法对于函数的形态十分敏感, 不论是在求解成功率方面还是在计算次数方面, 在不同函数之间的性能评测指标相差悬殊。

- (2) 对于 DE 算法而言,其在求解三个 UE 问题的过程中求解成功率都超过了 20%,这说明 DE 算法更有可能成功计算出不同形态连续最优问题的最优解。然而,DE 算法的待测函数平均运行次数较多。因此,DE 算法在寻找最优解概率比较大的目标函数上运行相对比较慢。
- (3) 对于 GA 算法而言,尽管依照前文经验 GA 算法的效果最劣,但是在执行其他算法的 UE 问题时,GA 算法并未体现出完全的劣势。由于 GA 算法能够更快地进入求解停滞状态,在待测函数平均执行次数方面,GA 算法表现最稳定。

### 4.3 对 UE (DE) 问题剖析

在本节及之后的两节,本文将对二维目标 UE 问题进行形态与求解数据上的分析。实验函数即选用 4.1 节三个目标函数,实验数据选用表 4.2,表 4.3 所对应的数据,目标是对实验结果进行更深层次的解释。

图 4.1 显示了 UE (DE) 问题目标函数的二维函数平面图像。其中图中横,纵坐标分别代表了自变量 $X_1$ 与 $X_2$ 的值,范围为 $[-1,1]$ ,在坐标 $(X_1, X_2)$ 处,颜色越浅,说明函数的值越小,在右侧的颜色条中可以找到颜色对应的数值。在图中有一点标明了红色的星形,代表了这个二维函数图像的最小值点。

图 4.2 显示了 UE (DE) 问题目标函数的二维函数立体图像。立体坐标的竖轴表示函数值。为方便函数最小值的观察分析,本文将立体坐标竖轴反转,即坐标点越靠上,坐标点所对应的 z 坐标值越小,对应的函数值也越小。立体坐标系中 x 轴与 y 轴也分别代表了自变量 $X_1$ 与 $X_2$ 的值。蓝色网格纹的平面代表了 UE (DE) 问题目标函数的二维对应关系。在图中有一个红色的圆点,代表了这个二维函数图像的最小值点,即蓝色网格的最高点。

从图 4.1 中容易看出,二维函数值呈现了一种类似关于直线 $X_1 = -X_2$ 对称的形态(但实际上通过函数形式上可看出函数并非关于此直线对称,存在一些偏差)。大体来说,距离直线 $X_1 = -X_2$ 越近的点,其函数值越小。在  $(0.5, -0.5)$  点附近处,在直线 $X_1 = -X_2$ 上函数有一个极大值,从而导致二维函数产生了红色星形点处与  $(1, -1)$  点附近两个极小值。然而,在求解目标函数的过程中,分析实验数据发现,三个算法所有进化算法个体都未陷入  $(1, -1)$  点附近的局部最优解。这说明三个算法在处理最优化问题时都有相当的跳出局部最优解,寻找全局最优解的能力。

从图 4.2 中可以进一步看出,立体函数图像呈现出“折曲面”的形态,其折线部分即位于 $X_1 = -X_2$ 直线附近。出现此情况是因为在函数表达式中运用的运算含有绝对值运

算，其中一个绝对值运算的自变量为 $X_1 + X_2$ ，这自然导致了函数在 $X_1 + X_2 > 0$ 处函数图像与 $X_1 + X_2 < 0$ 处函数图像的不一致性。

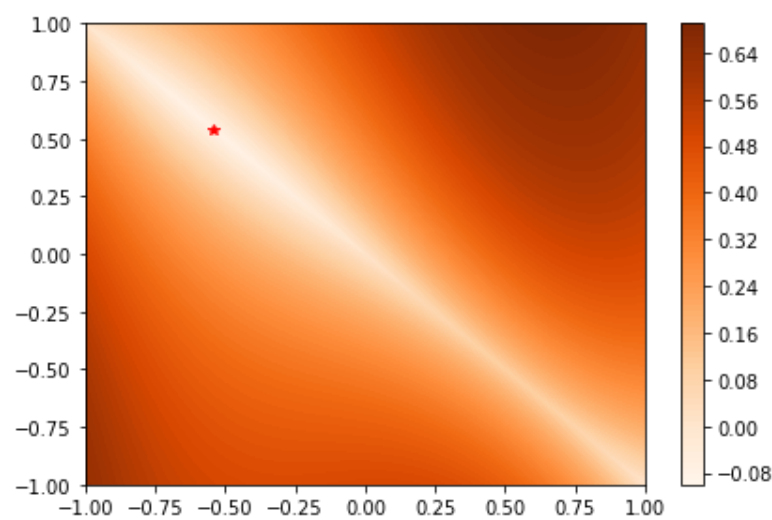


图 4.1 UE (DE) 二维目标函数图像——平面图

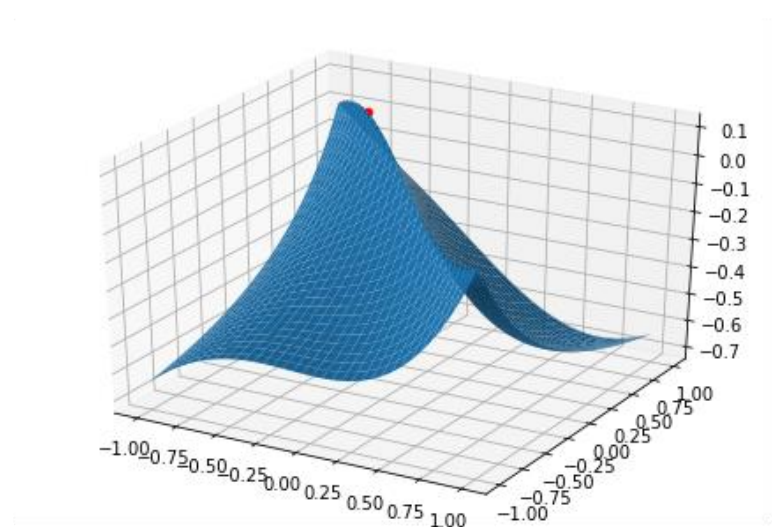


图 4.2 UE (DE) 二维目标函数图像——立体图

图 4.3 显示了三个算法在计算 UE (DE) 二维目标函数时计算值与最小值之间的误差随计算次数变化的关系。实验数据对应了表 4.3 中的数据，图像数据选取规则如下：对于算法 A 的第  $n$  次计算结果，选取算法 A 所有求解数据的第  $n$  次计算结果的中位数。对于算法某次求解的第  $n$  次计算没有实验数据的（即此求解的计算次数小于  $n$  的），若此次求解是成功的，以最小值准确度 $\delta_B = 10^{-10}$ 作为实验数据（在图中以绿色虚线表示

此值)。若此次求解不成功,以此次求解的最后一次计算结果作为实验数据。本章中的三个折线图仍然采用对数刻度显示。

由图 4.3 中三条曲线的趋势可看出,在拟合最初时,算法 DE 的表现并不优秀。在待测函数执行次数小于 30 次时,DE 算法的误差要大于 GA 算法与 PSO 算法。然而,GA 算法的计算在距离最小点仍然很远的地方就陷入了停滞,并在待测函数执行了 100 次左右时拟合完全陷入停滞。PSO 算法在待测函数执行 100 次左右时与最小值的误差减少速度开始减缓,并在待测函数执行 500 次左右时被 DE 算法反超。在待测函数执行 800 次左右时,PSO 算法的拟合完全陷入停滞。然而,对于 DE 算法来说,在误差达到 $\delta_B$ 之前,函数对数刻度图像几乎为直线,即计算的误差和计算次数之间几乎完全呈现了对数减小的关系。

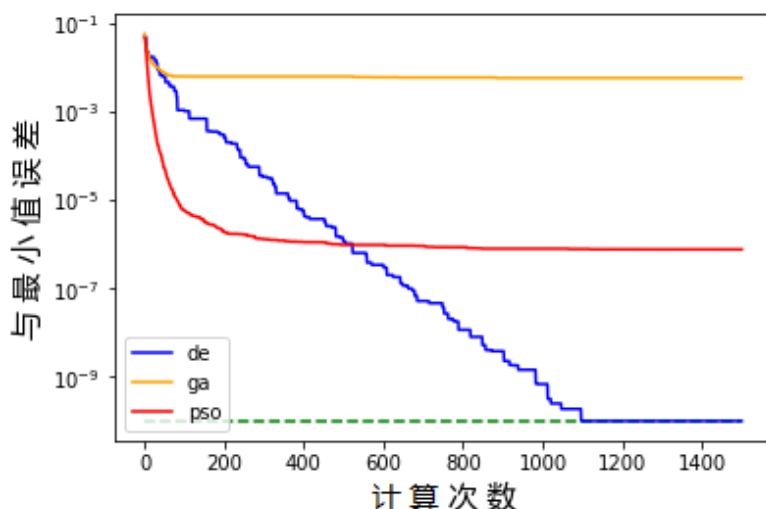


图 4.3 计算 UE (DE) 二维目标函数时  
计算值与最小值误差与  
待测函数计算次数之间的关系

DE 拟合 UE (DE) 目标问题性能优秀的原因可以从各算法的原理以及本文 UE (DE) 函数的特点得出。DE 算法的基本处理方法是:在种群中随机两个个体之间进行加权的差向量的运算,将其加到第三个个体上来产生新的向量,然后将新个体与原个体进行比较以决定这个个体是否更新。由于这种“向量差”的计算方式,当算法中种群的所有个体都在同一条直线上时,计算出的下一代个体便仍然在这条直线上。也就是说,DE 算法的种群习惯于呈现一种直线分布的形式。而对于其他两个算法而言,即便是某一代中

种群的所有个体都在同一条直线上，下一代中种群也会搜寻在这条直线之外是否有更优解。

对于 UE (DE) 目标函数而言，从图 4.1 中便可以看出，其较小值都分布在直线 $X_1 + X_2 = 0$ 上，而其最小值点也恰在这条直线上，为  $(-0.54148, 0.54148)$ 。DE 算法在求解目标问题时步骤可以大体描述如下：

- (1) 由于直线 $X_1 + X_2 = 0$ 上的点性能指标要远大于直线以外的点，DE 算法种群中所有个体首先拟合到直线 $X_1 + X_2 = 0$ 附近
- (2) 此时，DE 算法生成的新个体几乎都在直线 $X_1 + X_2 = 0$ 上，进而转化到寻找直线 $X_1 + X_2 = 0$ 中最优解的问题上。

因为求解的最终目的是对整个函数的求解，在别的算法还在寻找直线 $X_1 + X_2 = 0$ 之外的最优解时，DE 算法将精力放在了求解直线 $X_1 + X_2 = 0$ 最优解的问题上。因此，进行相同次的运算，DE 算法找到最优解的概率大大提高了，对于 UE (DE) 目标问题的求解性能也提高了。

这个解释也验证了前文 3.2 节得到的经验，即 DE 算法适合解决低维度问题。因为种群数量有限，DE 算法在求解高维问题时很有可能陷入一个低维的空间导致求解失败，所以 DE 算法在高维度问题下性能会大大降低。

#### 4.4 对 UE (GA) 问题剖析

图 4.4，图 4.5 分别显示了 UE (GA) 问题目标函数的平面图像和立体图像，图像生成方法与上一节所述相同。

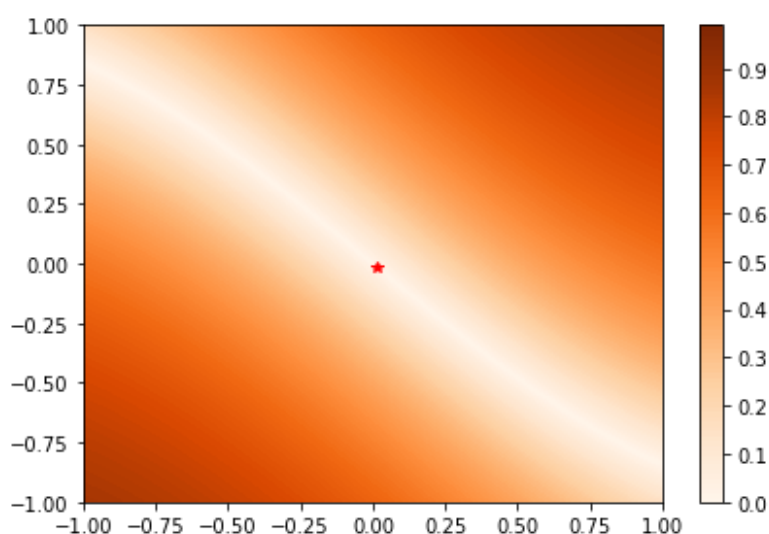


图 4.4 UE (GA) 二维目标函数图像——平面图

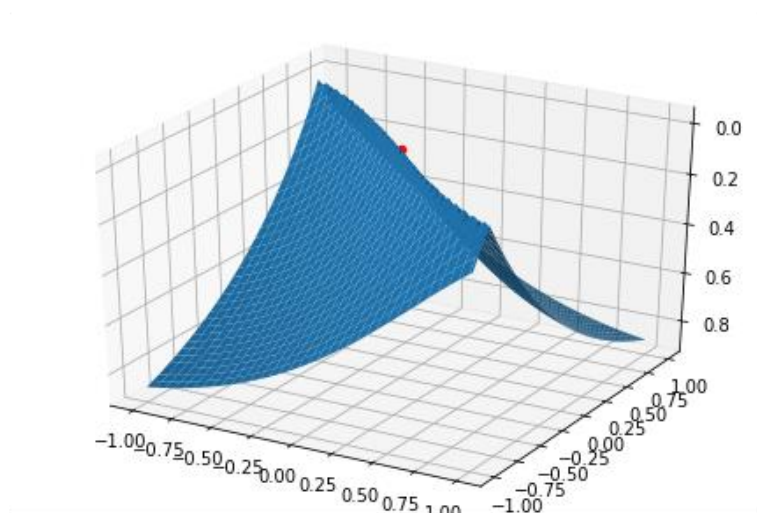


图 4.5 UE (GA) 二维目标函数图像——立体图

观察 UE (GA) 函数图像，其形态与 UE (DE) 问题的函数图像部分相似，然而二者的不同使得  $S_A$  中的三个算法运行两函数时性能相差悬殊：

对于正常的二维函数来说，函数最优解应该对应一个点。然而，结合 UE (GA) 的函数和图像分析，函数的最优解对应了一条线，方程为  $X_1 + \sin X_2 = 0$ 。这实际上使得算法中个体在函数最优解附近的概率增大了。

尽管函数在直线  $X_1 + X_2 = 0$  上的函数值非常小，但是函数最优解  $X_1 + \sin X_2 = 0$  实质上是一条曲线。两条线的相交处在定义域内只有点  $(0,0)$ 。因此，可以理解为曲线  $X_1 + \sin X_2 = 0$  的似直线形式对 DE 算法具有一定的误导作用，这使得 DE 算法在此问题上求解成功率反而下降了。

图 4.6 显示了三个算法在计算 UE (DE) 二维目标函数时误差随计算次数变化的关系，图像生成方法同上节。从图中可以得知，DE 算法一直是三个算法中计算误差最大的，且整个计算过程中 DE 算法的误差减小速度一直在减慢，并在第 820 次左右计算陷入了停滞。从最开始计算直到第 70 次计算左右，GA 算法与 PSO 算法计算的速度相差不多。然而，在此之后，PSO 算法意外地有一段明显的计算停滞。这段停滞拉开了两个算法之间误差的距离。最终，GA 算法比 PSO 算法提前拟合成功。

由本节和上节讨论可知，PSO 算法不适合处理折面问题。实际上，PSO 在种群拟合稳定后，种群中个体会以种群中心为中心振荡。而折面问题相当于在函数最优解的局部，其中一个维度（垂直折线方向）变化非常大，而另一维度（沿折线方向）变化非常小，这对振荡的个体接近最优解来说是非常不利的。

综合以上经验，对于二维函数来说，算法 **GA** 更适合求解最优解为曲线形态，整个函数为“折面”形态的函数问题。最优解为曲线形态导致最优解范围的拓宽对 **GA** 算法的性能有非常大的提升作用。

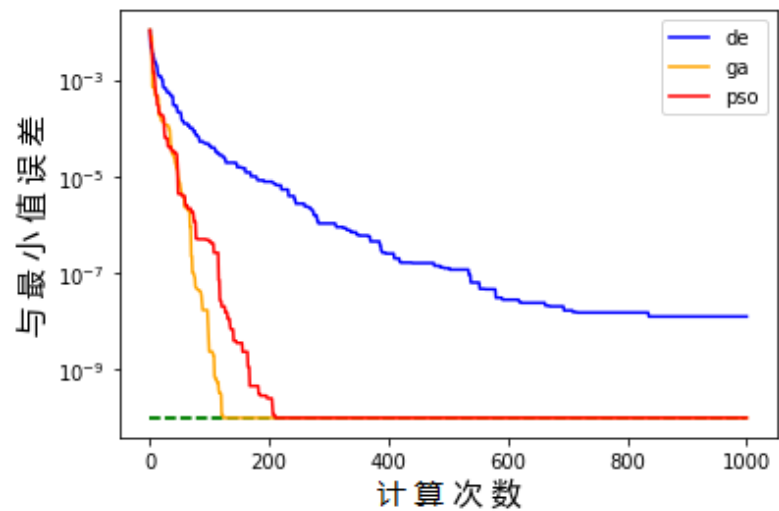


图 4.6 计算 UE (GA) 二维目标函数时  
计算值与最小值误差与  
待测函数计算次数之间的关系

4.5 对 UE (PSO) 问题剖析

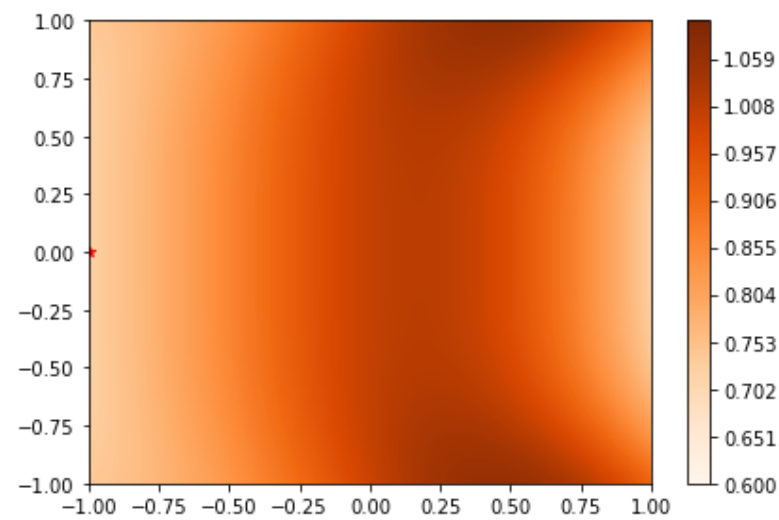


图 4.7 UE (PSO) 二维目标函数图像——平面图



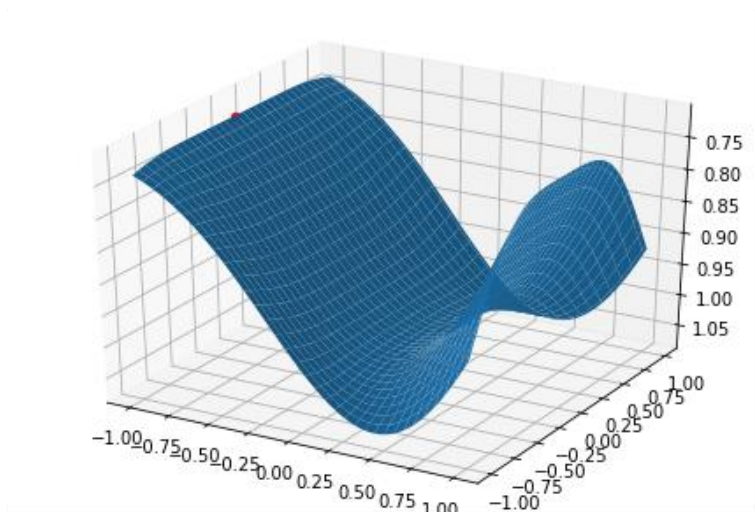


图 4.8 UE (PSO) 二维目标函数图像——立体图

图 4.7, 图 4.8 分别显示了 UE (PSO) 问题目标函数的平面图像和立体图像, 图像生成方法与前两节叙述相同。

UE (PSO) 目标函数相比其他两个 UE 函数在形态上相差很多。从立体图像上来说, 函数图像呈现了一个两侧向函数最小值方向弯曲的形态。从最优解位置方面来说, 此函数的最优解落在了 $(-1,0)$ 这个函数边界点。从最优解附近的函数图像来看, 整个直线 $X_1 = -1$ 上的函数值相差非常小。在 UE (PSO) 目标函数求解的程序中, 程序算得的最小值点为 $(-1, 0.0037044)$ , 分析目标函数函数式易知程序的真实最小值点为 $(-1, 0)$ 。然而, 距离大于 0.003 的两点之间的函数值相差还不到 $10^{-14}$ 。这说明在直线 $X_1 = -1$ 上, 中间有一大部分的区域是被本文程序所认可的最小值点。

图 4.9 显示了三个算法在计算 UE (PSO) 二维目标函数时误差随待测函数计算次数变化的关系, 图像生成方法同上两节。尽管 DE 算法和 GA 算法分别在第 300 次到第 400 次先后计算最优解成功, 但由于 PSO 算法仅对待测函数计算了平均不到两次就计算成功, 导致得到的 UE (PSO) 指标非常好。

PSO 算法有几个基本的规则: (1) 飞离最近的个体以避免碰撞 (2) 飞向群体的中心避免离群 (3) 飞向目标。飞离最近个体的规则, 使得 PSO 算法有一定概率飞到函数的定义域边界。加之函数在直线 $X_1 = -1$ 附近 $X_1$ 维度上偏导很大, 种群个体很容易演化到直线 $X_1 = -1$ 上。飞向群体中心原则, 使得在直线 $X_1 = -1$ 上的种群个体容易向函数图像中部即最优解点靠拢 (实际上, 个体演化到直线 $X_1 = -1$ 上就已经有一定的概率符合了本文求解成功的要求)。因此, PSO 算法在最优解在边界这一类问题上表现极好。



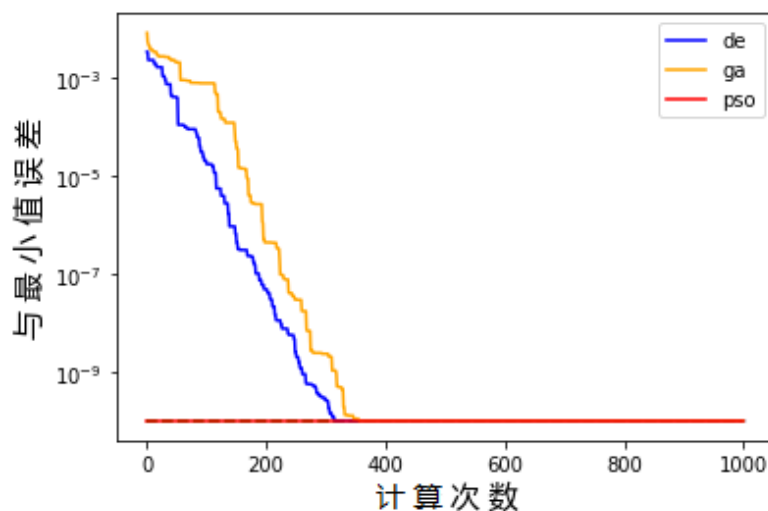


图 4.9 计算 UE (PSO) 二维目标函数时  
计算值与最小值误差与  
待测函数计算次数之间的关系

## 4.6 结论

本章前面部分已经对二维问题下各算法适合使用的问题的函数形式进行了一些验证与总结。本节依据本章前面的讨论，对上一章结论部分进行相应的补充：

本章证实了 **DE** 算法在函数维度比较少的问题上运行的性能比在函数维度比较多的问题上运行的性能好很多，且对于函数较优解完全分布在一条直线上的一类二维问题求解效果相对很好。本章证实了 **PSO** 算法在对最优解分布在定义域边界上的函数优化问题效果很好，但对最优解分布在定义域内部，且形态为折面，最优解在折线上的函数优化问题效果较差。**GA** 问题在函数最优解为一条线上的所有点时运行效果比较好。

## 结 论

本文提出了一种新式的解决 UE/UD 问题的方法。对于指定算法集 $\mathcal{S}_A$ ，本文生成一系列目标函数，使得算法集中算法对这一函数进行求解时，其中一个预先指定的算法的性能明显优于/劣于其他几个算法。相比前人的解决办法，本文采用了许多新的策略以适应此类问题：对于进化算法的性能度量，本文主要采用在其他参数相同的前提下以进化算法中每个个体执行待测函数的次数作为评价指标生成的基础，并辅以以下几个步骤生成完整的进化算法性能评价指标：设置双准确度参数，设置最大求解次数以及最大求解成功次数，设置最小命中率作为平滑因子，将 $A_T$ 与 $A_M$ 求比值作为 UE/UD 指标。对于测试问题的生成方法，本文使用基于遗传编程的函数表达式生成策略，辅以“抑制因子”与“函数再处理”等处理办法，并制定了遗传编程每代的交叉变异策略。本文对使用本文方法运行出的实验结果进行了详细地讨论，着重讨论每种算法所适合的目标函数的形式。本文对生成的目标函数进行了再剖析，使用折线图清晰反映各算法之间的差异，并且解释了绝大部分算法在其对应的 UE 目标问题上性能优秀的原因。

相比前人的解决办法，本文得到的性能指标结果更科学，稳定性更好。本文生成的目标问题函数形式更加多样化，更加易于书写且利于理解，且更具有普遍意义。本文将生成的函数以实验结果的形式列出，方便后人对本文实验进行复现。然而，必须承认本文对 UE/UD 问题解决办法仍有一些不足，如遗传编程会生成一些代表性差的个体，因此生成优秀个体通常需要更多时间；以运行次数为性能指标时没有考虑进化算法进化每一代本身所需的运行时间，等等。

## 参 考 文 献

- [1] D. Molina, F. Moreno-García and F. Herrera. Analysis among winners of different IEEE CEC competitions on real-parameters optimization: Is there always improvement?[C]// Proceedings of 2017 IEEE Congress on Evolutionary Computation (CEC), 5-8 June 2017, San Sebastian, 2017: 805-812
- [2] Lou Y, Yuen S Y, Chen G, et al. Evolving benchmark functions using kruskal-wallis test[C]// Proceedings of 2018 Genetic and Evolutionary Computation Conference, 15-19 July 2018, Kyoto, 2018: 1337-1341.
- [3] Lou Y, Yuen S Y. On constructing alternative benchmark suite for evolutionary algorithms[J]. Swarm and evolutionary computation, 2019, 44: 287-292.
- [4] Gallagher M, Yuan B. A general-purpose tunable landscape generator[J]. IEEE Transactions on Evolutionary Computation, 2006, 10(5): 590-603.
- [5] Storn R, Price K. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces[J]. Journal of Global Optimization, 1997, 11(4): 341-359.
- [6] Meneghini I R, Alves M A, Gasparcunha A, et al. Scalable and customizable benchmark problems for many-objective optimization[J]. Applied Soft Computing, 2020, 90: 106139.
- [7] Bailey D W. Genetic programming of development: A model[J]. Differentiation, 1986, 33(2): 89-100.
- [8] Holland J H. Genetic Algorithms and the Optimal Allocation of Trials[J]. SIAM Journal on Computing, 1973, 2(2): 88-105.
- [9] Boeringer D W, Werner D H. Particle swarm optimization versus genetic algorithms for phased array synthesis[J]. IEEE Transactions on Antennas and Propagation, 2004, 52(3): 771-779.
- [10] Fogel D B. An introduction to simulated evolutionary optimization[J]. IEEE Transactions on Neural Networks, 1994, 5(1): 3-14.
- [11] Back T, Hammel U, Schwefel H, et al. Evolutionary computation: comments on the history and current state[J]. IEEE Transactions on Evolutionary Computation, 1997, 1(1): 3-17.
- [12] Forrest S. Genetic algorithms: principles of natural selection applied to computation[J]. Science, 1993, 261(5123): 872-878.
- [13] Abdulrahman H, Khatib M. Classification of Retina Diseases from OCT using Genetic Programming[J]. International Journal of Computer Applications, 2020, 177(45): 41-46.

## 附录 A 实验参数符号表

参数符号	参数意义	参数值
$p_A$	进化算法中种群的个体数目（见 1.2 节）	50
$\delta_C$	双准确度参数之计算准确度，当两个计算值数值差小于 $\delta_C$ 时认为两计算值数值相等（见 1.3 节）	$10^{-11}$
$\delta_B$	双准确度参数之最小值准确度，当求解最小值与事先求得的最小值差在 $\delta_B$ 以内时认为求解完成（见 1.3 节）	$10^{-10}$
$n_s$	连续计算相等次数容忍度，单次求解时，当连续 $n_s$ 次待测函数运行前后结果仍“计算相等”且仍未求解完成时认为求解失败（见 1.3 节）	400
$n_V$	求解成功次数上限，每次计算函数综合评定指标时，求解成功的函数求解次数不超过 $n_V$ （见 1.4 节）	10
$n_T$	求解次数上限，每次计算函数综合评定指标时，函数总的求解次数不超过 $n_V$ （见 1.4 节）	200
$\alpha_M$	最小命中率，用于平滑实验结果，参与综合评定指标 $\bar{n} \times \frac{100\%}{\alpha + \alpha_M}$ 计算（见 1.4 节）	0.005
$k$	抑制因子，依据遗传编程树节点深度影响节点生成终止符的概率来控制树高（见 2.3 节）	0.1
$k_v$	变异概率因子，依据遗传编程树节点深度影响节点被选为变异节点的概率（见 2.5 节）	0.1
$k_i$	交叉概率因子，依据遗传编程树节点深度影响节点被选为交叉节点的概率（见 2.5 节）	0.1
$\alpha_v$	遗传编程树节点为终止符时生成变量的概率（见 2.2 节）	0.8
$\alpha_c$	遗传编程树节点为终止符时生成常量的概率， $\alpha_v + \alpha_c = 1$ （见 2.2 节）	0.2
$m$	遗传编程种群数量	40
$t$	演化次数	30

## 附录 B 遗传编程个体的演化策略 (m=40)

将要演化生成的种群个体依次编号为 1 到 40。对编号为前 23 的个体先进行演化，演化策略如下：（文中标号为个体编号，内容为个体演化策略）

- (1) 上一代中最优个体
- (2) 上一代中最优个体与上一代中次优个体子树进行交叉后生成的个体（即上一代中最优个体的一个随机节点替换为上一代中次优个体的随机子树）
- (3) 上一代中最优个体与上一代中次优个体子树进行交叉后生成的个体
- (4) 上一代中最优个体与上一代中次优个体子树进行交叉后生成的个体
- (5) 上一代中最优个体与上一代中第三优个体子树进行交叉后生成的个体
- (6) 上一代中最优个体与上一代中第三优个体子树进行交叉后生成的个体
- (7) 上一代中最优个体与上一代中第四优个体子树进行交叉后生成的个体
- (8) 上一代中最优个体通过变异后产生的个体
- (9) 上一代中最优个体通过变异后产生的个体
- (10) 上一代中次优个体
- (11) 上一代中次优个体与上一代中最优个体子树进行交叉后生成的个体
- (12) 上一代中次优个体与上一代中最优个体子树进行交叉后生成的个体
- (13) 上一代中次优个体与上一代中第三优个体子树进行交叉后生成的个体
- (14) 上一代中次优个体通过变异后产生的个体
- (15) 上一代中第三优个体
- (16) 上一代中第三优个体个体与上一代中最优子树进行交叉后生成的个体
- (17) 上一代中第三优个体个体与上一代中次优子树进行交叉后生成的个体
- (18) 上一代中第三优个体个体通过变异生成的个体
- (19) 上一代中第四优个体
- (20) 上一代中第四优个体与上一代中最优子树进行交叉后生成的个体
- (21) 上一代中第四优个体个体通过变异生成的个体
- (22) 上一代中第五优个体
- (23) 上一代中第五优个体与上一代中最优子树进行交叉后生成的个体

此后，以上一代中性能指标排名第 6 到第 20 的个体为主体，与上一代中性能指标排名前 30 的个体中的随机子树进行交叉，共生成 15 个个体，对应本代的编号为 24 到 38。

最后，生成新随机树 2 个（模拟外来个体），作为本代的 39 号和 40 号个体。

## 附录 C 实验结果一部分原输出数据 (m=40, t=30)

此表为实验结果一的输出部分, 其中对于输出符号  $\text{var}[i]$ , 对应的公式名称为  $X_{i+1}$ , \*为乘号。

由于实验结果过长, 对于每个实验, 本文仅列出最后五代函数的最优个体与各实验中最后一代中排名前五的个体。

### (1) n=2 时 UE(DE)问题:

第 26 到 30 代函数的最优个体 (按演化代数递增顺序列出, 下同):

- ①  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\text{var}[1]))$
- ②  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\text{var}[1]))$
- ③  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\text{var}[1]))$
- ④  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\text{var}[1]))$
- ⑤  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\sin(\text{var}[0])))$

最后一代函数的排名前五个体 (按排名顺序列出, 下同):

- ①  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\sin(\text{var}[0])))$
- ②  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\sin(\sin((\text{var}[1]+(-0.41424336297007364))))))$
- ③  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\text{var}[1]))$
- ④  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\text{var}[1]))$
- ⑤  $(\log(\text{abs}(\sin(\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1))))+1)+\sin(\text{var}[1]))$

### (2) n=2 时 UE(GA)问题:

第 26 到 30 代函数的最优个体:

- ①  $\sin(\log(\text{abs}((\sin(\text{var}[1])+\sin((\cos(\log(\text{abs}(\text{var}[0])+1))*\log(\text{abs}(\text{var}[0])+1))))+1))$
- ②  $\sin(\log(\text{abs}((\text{var}[1]+\sin((\cos(\log(\text{abs}(\text{var}[0])+1))*\sin(\text{var}[0])))+1))$
- ③  $\sin(\sin(\log(\text{abs}(\sin((\text{var}[0]+\sin((\text{var}[1]*(0.4018368171615318))))+1))))$
- ④  $\sin(\log(\text{abs}((\text{var}[1]+\sin((\cos(\log(\text{abs}(\text{var}[0])+1))*\sin(\text{var}[0])))+1))$
- ⑤  $\sin(\log(\text{abs}((\text{var}[1]+\sin(\text{var}[0]))+1))$

最后一代函数的排名前五个体:

- ①  $\sin(\log(\text{abs}((\text{var}[1]+\sin(\text{var}[0]))+1))$
- ②  $\sin(\log(\text{abs}((\text{var}[1]+\sin((\cos(\log(\text{abs}(\text{var}[0])+1))*\sin(\text{var}[0])))+1))$
- ③  $\sin(\log(\text{abs}(((\text{var}[0]+\text{var}[0])+\sin(\text{var}[1]))+1))$

- ④  $\sin(\log(\text{abs}((\text{var}[1]+(\log(\text{abs}((\text{var}[0]*\text{var}[0]))+1)*\cos(\cos(\log(\text{abs}(\text{var}[0])+1))))))$   
 $+1))$
- ⑤  $\sin(\log(\text{abs}(((\text{var}[0]+\text{var}[0])+\sin((\text{var}[1]+\text{var}[0])))+1))$

(3) n=2 时 UE(PSO)问题:

第 26 到 30 代函数的最优个体:

- ①  $\sin((\cos((0.23171727502870929))+\cos((\text{var}[0]+\cos(\sin((\sin(\text{var}[1])*\text{var}[1]))))))$
- ②  $\sin((\cos((0.23171727502870929))+\cos((\text{var}[0]+\cos(\sin((\sin(\text{var}[1])*\text{var}[1]))))))$
- ③  $\sin((\cos((\text{var}[0]*\cos(((\text{var}[1]*\sin(\text{var}[1]))*\text{var}[1]))))+\cos(\text{var}[0]))$
- ④  $\sin((\cos((\text{var}[0]*\cos(((\text{var}[1]*\sin(\text{var}[1]))*\text{var}[1]))))+\cos(\text{var}[0]))$
- ⑤  $\sin((\cos((0.23171727502870929))+\cos((\text{var}[0]+\cos(\sin((\sin(\text{var}[1])*\text{var}[1]))))))$

最后一代函数的排名前五个体:

- ①  $\sin((\cos((0.23171727502870929))+\cos((\text{var}[0]+\cos(\sin((\sin(\text{var}[1])*\text{var}[1]))))))$
- ②  $\sin((\cos((\text{var}[0]*\cos(((\text{var}[1]*\text{var}[1])* \sin(\sin(\text{var}[1]))))))+\cos((\text{var}[0]+(0.1020612$   
 $7848272989))))$
- ③  $\sin((\cos((\text{var}[0]*\cos(((\text{var}[1]*\sin(\text{var}[1]))*\text{var}[1]))))+\cos(\text{var}[0]))$
- ④  $\sin((\cos((0.23171727502870929))+\cos((\text{var}[0]+\cos(\sin((\sin(\text{var}[1])*\text{var}[1]))))))$
- ⑤  $\sin((\cos(\cos(\text{var}[0]))+\cos((\text{var}[0]+\cos((\log(\text{abs}(\text{var}[1])+1)*\text{var}[1]))))))$

(4) n=10 时 UE(DE)问题:

第 26 到 30 代函数的最优个体:

- ①  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}(\text{var}[9])+1))+(-0.5352565680362621))$
- ②  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}(\text{var}[9])+1))+(-0.5352565680362621))$
- ③  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}(\text{var}[9])+1))+((\log(\text{abs}((($   
 $0.9338181072730535)+\text{var}[3]))+1)*\log(\text{abs}(\cos(\text{var}[4])+1))*(-$   
 $0.6221569222962497))))$
- ④  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}((\log(\text{abs}(\log(\text{abs}((\text{var}[6]+\text{var}[7]))+1))+1)*\sin(\log(\text{abs}(($   
 $0.45549102932643176))+1))))+1))+(-0.5352565680362621))$
- ⑤  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}((\log(\text{abs}(\log(\text{abs}((\text{var}[6]+\text{var}[7]))+1))+1)*\sin(\log(\text{abs}(($   
 $0.45549102932643176))+1))))+1))+(-0.5352565680362621))$

最后一代函数的排名前五个体:

- ①  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}((\log(\text{abs}(\log(\text{abs}((\text{var}[6]+\text{var}[7]))+1))+1)*\sin(\log(\text{abs}(($   
 $0.45549102932643176))+1))))+1))+(-0.5352565680362621))$

- ②  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}(\text{var}[9])+1)+((\log(\text{abs}((-0.9338181072730535)+\text{var}[3]))+1)*\log(\text{abs}(\cos(\text{var}[4])+1))*(-0.6221569222962497))))$
- ③  $((\cos(\text{var}[6])* \sin(\sin(\sin((-2.424503698781985))))+(\log(\text{abs}(\text{var}[9])+1)+(\cos((\text{var}[6]+\cos(\log(\text{abs}(\text{var}[3])+1))))+\text{var}[0])))$
- ④  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}(\text{var}[9])+1)+((\text{var}[4]*\text{var}[4])*((0.9618146715914495)*\log(\text{abs}((-3.7186746272175646))+1))))))$
- ⑤  $(\cos(\cos(\text{var}[6]))+(\log(\text{abs}(\text{var}[9])+1)+(-0.5352565680362621)))$

(5) n=10 时 UE(GA)问题:

第 26 到 30 代函数的最优个体:

- ①  $\log(\text{abs}(((\log(\text{abs}(\text{var}[0])+1)+((\text{var}[4]*\sin(\text{var}[7]))*\text{var}[5]))+(\log(\text{abs}((\sin(\sin(\text{var}[3]))*\sin((\text{var}[5]*\text{var}[7])))+1)*\cos(\text{var}[3])))+1))$
- ②  $\log(\text{abs}(((\log(\text{abs}(\text{var}[0])+1)+((\text{var}[1]*\text{var}[2])* \text{var}[5]))+(\log(\text{abs}(((3.182369130724782)*\sin((\text{var}[5]*\sin(\text{var}[8])))+1)*\sin(\text{var}[3])))+1))$
- ③  $\log(\text{abs}((((\log(\text{abs}(\text{var}[7])+1)+\sin(\text{var}[3]))*\sin(((\text{var}[7]+\sin(\text{var}[1]))*(-0.704978535850138)+(\text{var}[1]*\text{var}[8])))+((\text{var}[9]+\text{var}[8]+\text{var}[6])))+1))$
- ④  $\log(\text{abs}(((\log(\text{abs}(\text{var}[9])+1)+(((\text{var}[2]+((0.3705594895327738)*\text{var}[0]))*\text{var}[0])* \text{var}[5]))+(\log(\text{abs}(\log(\text{abs}(\sin(\text{var}[7]))+1))+1)*\sin(\log(\text{abs}(\text{var}[3])+1)))))+1))$
- ⑤  $\log(\text{abs}((((\sin(\text{var}[7]))*(\sin(\text{var}[0])+(\text{var}[1]+\cos(\text{var}[9]))))*(\log(\text{abs}((-0.8564779454182956))+1)*\text{var}[8]))+(\text{var}[0]+(((\text{var}[6]*\text{var}[0])+((\text{var}[9]+\text{var}[0])+\text{var}[2]))*\text{var}[3])*((\text{var}[5]+\cos(\text{var}[8]))+(\text{var}[0]+\text{var}[5])))))))+1))$

最后一代函数的排名前五个体:

- ①  $\log(\text{abs}((((\sin(\text{var}[7]))*(\sin(\text{var}[0])+(\text{var}[1]+\cos(\text{var}[9]))))*(\log(\text{abs}((-0.8564779454182956))+1)*\text{var}[8]))+(\text{var}[0]+(((\text{var}[6]*\text{var}[0])+((\text{var}[9]+\text{var}[0])+\text{var}[2]))*\text{var}[3])*((\text{var}[5]+\cos(\text{var}[8]))+(\text{var}[0]+\text{var}[5])))))))+1))$
- ②  $\log(\text{abs}(((\log(\text{abs}(\text{var}[9])+1)+(((\text{var}[3]*(\text{var}[7]*\text{var}[4]))*\text{var}[0])*((\text{var}[2]*\text{var}[0])* \text{var}[8])* \sin(\cos(\cos(\text{var}[9])))))))+\sin(((\text{var}[1]+\text{var}[2])* \text{var}[1])))+1))$
- ③  $\log(\text{abs}((\sin(((\text{var}[3]+\cos(\text{var}[5]))*(\text{var}[4]*\log(\text{abs}(\log(\text{abs}(\text{var}[6])+1))+1)))))+(((2.018540322518934)+(\text{var}[9]*\log(\text{abs}(\text{var}[0])+1))* \text{var}[2])))+1))$
- ④  $\log(\text{abs}(((\log(\text{abs}(\text{var}[0])+1)+(((\text{var}[7]+\text{var}[6])* \text{var}[2])* \text{var}[5]))+(\cos((\log(\text{abs}(\text{var}[7])+1)* \text{var}[9]))*(\text{var}[4]*\text{var}[9])))+1))$



- ⑤  $\log(\text{abs}(((\text{var}[8]+((\text{var}[2]+(-0.6571821721530973))\cdot\text{var}[0])\cdot\text{var}[5]))+(\log(\text{abs}(\log(\text{abs}(\sin(\text{var}[7]))+1))+1)\cdot\sin(\log(\text{abs}(\text{var}[3])+1)))))+1)$

(6) n=10 时 UE(PSO)问题:

第 26 到 30 代函数的最优个体:

- ①  $((\text{var}[7]+\sin(\cos(((\text{var}[6]\cdot\text{var}[0])\cdot(1.975338108318378))))\cdot\text{var}[3])$   
 ②  $((\text{var}[5]\cdot(\log(\text{abs}(\text{var}[3])+1)+(2.734183794868712))\cdot(\text{var}[9]+\text{var}[0]))$   
 ③  $((\text{var}[7]+(\sin(\cos(\sin(((3.818187698271945)+\text{var}[3]))\cdot\sin((\text{var}[2]\cdot\text{var}[6]))\cdot\text{var}[3])$   
 ④  $((\text{var}[7]+(\sin(\cos(\sin((\text{var}[3]+\cos(\cos(\text{var}[2]))\cdot\sin((\text{var}[2]\cdot\text{var}[6]))\cdot\text{var}[3])$   
 ⑤  $((\text{var}[5]\cdot(\text{var}[5]+\text{var}[3]))\cdot(\text{var}[9]+(\cos(\text{var}[2])\cdot\cos((0.2464731912488536))))$

最后一代函数的排名前五个体:

- ①  $((\text{var}[5]\cdot(\text{var}[5]+\text{var}[3]))\cdot(\text{var}[9]+(\cos(\text{var}[2])\cdot\cos((0.2464731912488536))))$   
 ②  $((\text{var}[5]\cdot(\text{var}[5]+\text{var}[3]))\cdot(\text{var}[9]+(\cos(\text{var}[2])\cdot\cos(\text{var}[9]))))$   
 ③  $((\text{var}[7]+(\sin(\cos(\sin((\text{var}[3]+\cos(\cos(\text{var}[2]))\cdot\sin((\text{var}[2]\cdot\text{var}[6]))\cdot\text{var}[3])$   
 ④  $((\text{var}[0]+\sin(\cos(\log(\text{abs}(\text{var}[6])+1))))\cdot\text{var}[7])$   
 ⑤  $((\text{var}[7]+\sin(\cos(((1.1619773759076206)+\sin(((3.3365202448434834)+(\text{var}[7]\cdot(\text{var}[7]\cdot\text{var}[8]))\cdot\text{var}[3])$

(7) n=2 时 UD(DE)问题:

第 26 到 30 代函数的最优个体:

- ①  $\log(\text{abs}(((\text{var}[1]\cdot((\text{var}[0]\cdot\sin(\text{var}[1]))\cdot\text{var}[0]))+\sin((-2.8648192152939274))))+1)$   
 ②  $\log(\text{abs}((\sin(((\text{var}[1]\cdot\log(\text{abs}(\text{var}[0])+1))\cdot\text{var}[1]))+\sin((-2.8648192152939274))))+1)$   
 ③  $\log(\text{abs}((\log(\text{abs}(\log(\text{abs}(\cos(\text{var}[1]))+1))+1)\cdot\cos(((\text{var}[0]\cdot(\cos(\text{var}[1]))\cdot((\text{var}[0]+\text{var}[0])+\text{var}[1]))+\cos(\text{var}[0])))+1)$   
 ④  $\log(\text{abs}((\log(\text{abs}((2.6541784681304565))+1)\cdot(\cos((\cos((\log(\text{abs}(\text{var}[0])+1)+\cos(2.81698415278971))))\cdot(-4.5768825786391085)))+(\log(\text{abs}((\text{var}[1]\cdot\text{var}[0]))+1)+\sin(\sin(\log(\text{abs}(\text{var}[1])+1))))+1)$   
 ⑤  $\log(\text{abs}(((\cos((\text{var}[1]+(\sin(((\text{var}[1]\cdot\text{var}[1])+(0.4651794212380645))))\cdot\sin(\text{var}[1]))+(\log(\text{abs}(\text{var}[1])+1)\cdot(\log(\text{abs}((\cos(\text{var}[1])\cdot\text{var}[0]))+1)\cdot\sin((2.151144624681141)))))+\sin((-2.8648192152939274))))+1)$

最后一代函数的排名前五个体：

- ①  $\log(\text{abs}(((\cos((\text{var}[1])+(\sin(((\text{var}[1]*\text{var}[1])+(0.4651794212380645))))*\sin(\text{var}[1])))))+(\log(\text{abs}(\text{var}[1])+1)*(\log(\text{abs}((\cos(\text{var}[1])*\text{var}[0]))+1)*\sin((2.151144624681141))))))+\sin((-2.8648192152939274))))+1)$
- ②  $\log(\text{abs}((\log(\text{abs}((2.6541784681304565))+1)*(\cos((\cos((\log(\text{abs}(\text{var}[0])+1)+\cos((2.81698415278971))))*(-4.5768825786391085))+(\log(\text{abs}((\text{var}[1]*\text{var}[0]))+1)+\sin(\sin(\log(\text{abs}(\text{var}[1])+1))))))+1)$
- ③  $\log(\text{abs}((\log(\text{abs}(\log(\text{abs}(\cos(\text{var}[1]))+1))+1)*\cos(((\text{var}[0]*(\cos(\text{var}[1]))*(\text{var}[0]+\text{var}[0])+\text{var}[1])))+\cos(\text{var}[0]))))+1)$
- ④  $\log(\text{abs}(((\text{var}[1]*(\text{var}[0]*\cos(\text{var}[0])))+(-0.4059741366925766))+1)$
- ⑤  $\log(\text{abs}(\log(\text{abs}(\cos(((\text{var}[0])+(-0.027727584182626268))+\sin(\text{var}[0]))*\text{var}[1]))+1))+1)$

(8) n=2 时 UD(GA)问题：

第 26 到 30 代函数的最优个体：

- ①  $((\log(\text{abs}((\text{var}[1]+(\text{var}[0]*\text{var}[0])))+1)+\log(\text{abs}(\cos(\text{var}[0]))+1))+\log(\text{abs}((\cos((\cos((2.9008575572386097))+\cos(\text{var}[0])))*\sin(\text{var}[1])))+1))$
- ②  $((\log(\text{abs}((\text{var}[1]+(\text{var}[0]*\text{var}[0])))+1)+\log(\text{abs}(\cos(\text{var}[0]))+1))+\log(\text{abs}((\cos((\cos((2.9008575572386097))+\cos(\text{var}[0])))*\sin(\text{var}[1])))+1))$
- ③  $((\log(\text{abs}((\text{var}[1]+(\text{var}[0]*\text{var}[0])))+1)+\log(\text{abs}(\cos(\text{var}[0]))+1))+\log(\text{abs}((\cos((\cos((2.9008575572386097))+\cos(\text{var}[0])))*\sin(\text{var}[1])))+1))$
- ④  $((\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1)+\log(\text{abs}(\cos(\cos(((\text{var}[1]+\log(\text{abs}(\text{var}[1])+1))*\cos(\text{var}[0]))))+1))+\log(\text{abs}((\sin(\text{var}[0])* \cos((-0.7165480211984033)+\cos(\sin((-3.6327579823983243))))))+1))$
- ⑤  $((\log(\text{abs}((\text{var}[1]+(\text{var}[0]*\text{var}[0])))+1)+\log(\text{abs}(\cos(\text{var}[0]))+1))+\log(\text{abs}((\cos((\cos((2.9008575572386097))+\cos(\text{var}[0])))*\sin(\text{var}[1])))+1))$

最后一代函数的排名前五个体：

- ①  $((\log(\text{abs}((\text{var}[1]+(\text{var}[0]*\text{var}[0])))+1)+\log(\text{abs}(\cos(\text{var}[0]))+1))+\log(\text{abs}((\cos((\cos((2.9008575572386097))+\cos(\text{var}[0])))*\sin(\text{var}[1])))+1))$
- ②  $((\log(\text{abs}((\text{var}[0]+\text{var}[1]))+1)+\log(\text{abs}(\cos(\cos(((\text{var}[1]+\log(\text{abs}(\text{var}[1])+1))*\cos(\text{var}[0]))))+1))+\log(\text{abs}((\sin(\text{var}[0])* \cos((-0.7165480211984033)+\cos(\sin((-3.6327579823983243))))))+1))$
- ③  $((\log(\text{abs}((\text{var}[1]+(\text{var}[0]*\cos((\text{var}[1]*\text{var}[0]))))+1)+\log(\text{abs}(\cos(\text{var}[0]))+1))+\log(\text{abs}((\cos(\text{var}[0])* \sin(\log(\text{abs}(\text{var}[0])+1))))+1))$

- ④  $\log(\text{abs}((\log(\text{abs}(((\sin(\text{var}[1])+\sin(\sin(\text{var}[0]))) * \text{var}[1]))+1) * (((0.39039655861634764)+\sin((\text{var}[1]+\text{var}[0])))+\text{var}[0])))+1)$
- ⑤  $((\log(\text{abs}((\text{var}[1]+\text{var}[1]))+1)+\log(\text{abs}(\cos(\text{var}[0]))+1))+\log(\text{abs}(((\sin(\text{var}[0])+\sin(\text{var}[0])+\sin(\text{var}[1])) * (-2.1991303813747995))))+1))$

(9) n=2 时 UD(PSO)问题:

第 26 到 30 代函数的最优个体:

- ①  $\log(\text{abs}((((\text{var}[1] * \sin(\text{var}[1])) + \log(\text{abs}(\log(\text{abs}((\log(\text{abs}(\sin(\text{var}[1]))+1)+((-0.45163034800288043) * \text{var}[0])))+1))+1)) * \sin(\cos((\text{var}[1] * \text{var}[0])))+1)$
- ②  $\log(\text{abs}((((\text{var}[1] * \sin(\text{var}[1])) + \log(\text{abs}(\log(\text{abs}((\log(\text{abs}(\sin(\text{var}[1]))+1)+((-0.45163034800288043) * \text{var}[0])))+1))+1)) * \sin(\cos((\text{var}[1] * \text{var}[0])))+1)$
- ③  $(\sin(\log(\text{abs}((((((\cos(\text{var}[1])+\text{var}[1])+\text{var}[0])+\cos(\log(\text{abs}(\text{var}[0])+1))) * \cos(\text{var}[1])) * (2.923885088677733))))+1))+\sin(\sin(\text{var}[0])))$
- ④  $\log(\text{abs}((((\text{var}[1] * \sin(\text{var}[1])) + \log(\text{abs}(\log(\text{abs}((\log(\text{abs}(\sin(\text{var}[1]))+1)+((-0.45163034800288043) * \text{var}[0])))+1))+1)) * \sin(\cos((\text{var}[1] * \text{var}[0])))+1)$
- ⑤  $(\sin(\text{var}[1])+\sin(\log(\text{abs}(((\sin((\text{var}[0]+\text{var}[0]))+\text{var}[1]) * (1.8838226050802431))))+1)))$

最后一代函数的排名前五个体:

- ①  $(\sin(\text{var}[1])+\sin(\log(\text{abs}(((\sin((\text{var}[0]+\text{var}[0]))+\text{var}[1]) * (1.8838226050802431))))+1)))$
- ②  $\log(\text{abs}((((\text{var}[1] * \sin(\text{var}[1])) + \log(\text{abs}(\log(\text{abs}((\log(\text{abs}(\sin(\text{var}[1]))+1)+((-0.45163034800288043) * \text{var}[0])))+1))+1)) * \sin(\cos((\text{var}[1] * \text{var}[0])))+1)$
- ③  $(\sin(\log(\text{abs}((((((\cos(\text{var}[1])+\text{var}[1])+\text{var}[0])+\cos(\log(\text{abs}(\text{var}[0])+1))) * \cos(\text{var}[1])) * (2.923885088677733))))+1))+\sin(\sin(\text{var}[0])))$
- ④  $\log(\text{abs}((((\text{var}[1] * \sin(\text{var}[1])) + \log(\text{abs}(\sin((\text{var}[0]+\sin(\log(\text{abs}(\log(\text{abs}(\text{var}[1])+1)+1))))+1))) * \sin(\cos((\text{var}[1] * (1.2764716674800374)))))+1)$
- ⑤  $\log(\text{abs}((((\text{var}[1] * \sin(\text{var}[1])) + \log(\text{abs}(\log(\text{abs}((\log(\text{abs}(\text{var}[1])+1)+\text{var}[0]))+1))+1)) * \sin(\cos(\text{var}[1])))+1)$

## 修改记录

### 第一次修改记录:

第 5 页 2.4 前两自然段

#### 修改前:

通常情况下, 函数树生成后其对应的函数可以用来进行最优值的求解。然而, 这样生成的函数有一个问题: 由于遗传编程的随机性也伴随着不稳定性, 函数的最小值很可能落在函数定义域的边界, 即函数图像可能会“过于倾斜”。在一些算法(如 PSO)中, 由于算法具有“聚合性”的策略, 在生成初始种群时, 更倾向于生成自变量位于定义域边界的个体。虽然本文可以保证在定义域内函数有最小值, 但是当最小值位于图像边界的时候, 这些算法的初始种群会更接近函数的最小值点, 这样会极大地这些算法加快函数拟合的速度, 在评定时, 这些算法就会在这些“过于倾斜”的函数得到异常好的性能指标。

一方面, 如果只考虑将拟合迭代时的快慢作为各算法的性能, 将通过这种函数计算出的各算法的性能指标进行比较在某种程度上来说有失公正。但另一方面, S\_A 中各算法生成初始种群方法属于算法的一部分, 算法生成初始种群接近函数最小点, 从而能够快速找到函数的最小值, 也是算法在计算该函数时性能较高的一种体现。综合了上述两种意见, 本文对函数做了简单的处理。处理后的函数最小值更可能落在定义域的内部, 但也存在可能性最小值仍落在函数的边界。

#### 修改后:

通常情况下, 函数树生成后其对应的函数可以用来进行最优值的求解。然而, 这样生成的函数有一个问题: 由于遗传编程的随机性也伴随着不稳定性, 函数的最小值很可能落在函数定义域的边界, 即函数图像可能会“过于倾斜”。由经验得知, 在一些算法(如 PSO)中, 对于这样的函数拟合的速度过快。在评定时, 这些算法会在这些“过于倾斜”的函数得到异常好的性能指标。尽管遗传编程算法产生此类函数本身没有什么问题, 但生成的倾斜函数过多将会使得那些在倾斜函数上表现不好的算法找到性能指标好的目标函数的概率非常小, 时间过长。

因此, 本文对已经生成的个体函数做了简单的处理。处理后的函数最小值更可能落在定义域的内部, 但也存在可能性最小值仍落在函数的边界, 以此在保证生成函数多样性的前提下解决这个问题。

### 第二次修改记录:

修改了本文表格中数据精确度不统一的问题

**第三次修改记录：**

修改了对齐方式、图表跨页等问题

**第四次修改记录：**

修改论文关键词

**修改前：**遗传编程；最优化问题；进化算法；性能指标

**修改后：**遗传编程；连续优化问题；进化算法；性能指标；目标函数

**第五次修改记录：**

修改页眉格式问题

**第六次修改记录：**

修改公式为右对齐

修改参考文献格式

**第七次修改记录：**

修改每章第一节标题

**修改前：**引言


**修改后：**本章简介

按照模板标准修改第 24 页，第 15 页列表格式

修改参考文献格式为两端对齐

修改英文括号格式，括号两端留出空格。

记录人（签字）：张金哲

指导教师（签字）：

## 致 谢

本论文历时约五个月完成。首先感谢我的导师任志磊老师，在论文写作初期引导我明确了论文题目的方向。本文内容多为自主实验与总结，在整个论文撰写过程中，随着论文篇幅的增加，我也越发明白本文实验内容的重要性和实际价值，而这些要素成为了我努力进行实验，努力完成本论文的重要动力。

然而，在本次论文撰写与修改过程中我也感受到了自己的不足，在论文的格式规范方面，数据采集方面以及某些论文的措词仍存在一些不严谨的地方。再次感谢任志磊老师对我的论文进行再三指导。与此同时，感谢答辩组老师王雷老师，马瑞新老师和卢炳先老师对我的论文提出的指导意见。老师们的评阅让我感受到写论文是一件细致而长久的工作，需要耐心认真地对待这件事情。

同时，对本文撰写过程中参考的所有文献作者表示感谢。前人的工作对于本文具有极大的参考价值。