一个测量装置在大规模制造中的标定问题

522031910677　杨锦畅

日期：2024 年 4 月 7 日

摘要：本文提出了一个解决大规模制造中非线性元件标定问题的方法。采用改良的双种群遗传算法，将传统的单向个体流动转变为双向优质个体交流，结合自适应参数调整解决成本和生产效率优化问题。引入基于 Bayes 公式干预的变异概率控制方法，通过赋予基因片段不同的变异率来提高优质个体生成概率。同时，应用多种优化策略提升代码效率，为解决相似问题提供了可借鉴的技巧。此外。本文对遗传算法中轮盘赌、锦标赛、截断、随机遍历四种不同的选择算法进行了对比实验和讨论。

关键词：双种群遗传算法；遗传参数自适应；选择算法比较

一、 模型假设

1. **虽然不同传感器的性质有一定随机性，但存在一种规律能够描述整体的 $V-T$ 走势**。如果每组数据的走势均为随机（如温度传感器的温敏器件不同），则分析测试点的位置没有意义。
2. **题目给出的成本可以作为真实成本的评价依据**。其他可能的成本可以被忽略。
3. **题目所给数据均为精确数据，可以作为模型判断依据和训练根据**。

二、 符号说明

符号	意义
N	种群规模
X, Y	种群总体 X 和 Y
$f(x_i)$	第 i 个个体的适应度
P	概率参数向量
$s_{i,j}$	第 i 个个体，第 j 个待测温度点的误差成本
$T_{i,j}$	第 i 个个体，第 j 个待测温度点的精确温度
$\hat{T_{i,j}}$	第 i 个个体，第 j 个待测温度点的预测温度

注：以上为本论文中涉及的重要通用符号，对于少量具体问题中的具体符号会在使用时进行解释。

三、 问题分析

本题是一个典型的组合优化问题，需要从 90 个可行的温度点中找出有限个点进行标定，使得成本最优化。由于目标单一，且解的总集有限，故考虑采用遗传算法进行问题求解。

针对该解决方案，需要建立的数学模型有：

- **测定点集模型**，在此例中将其看作一个由 90 位二进制数组成的基因序列，用 0/1 表示该温度是否进行标定。
- **成本评价模型**，在此例中由题目给出，对每个测定点集可以计算平均成本，视作其适应度。

进行成本计算前，需要得到完整的温度-电压方程，而标定所直接得到的数据是若干个离散的点。因此，需要进行插值操作。根据对数据的初步分析，考虑采用三次样条插值方法补全未测定的温度点电压数据。

四、 模型建立与求解

4.1 简单遗传算法

4.1.1 基本思想

遗传算法 (Genetic Algorithm, GA) 是一种经典的仿生算法，通过矩阵模拟种群，而个体对应的向量则对应着一组解。在此题中，使用长度为 90 的 logical 向量表示一个解，其中值为 true 则表示该温度点被选择，反之表示其不被选择。在简单遗传算法中，对于每一代种群，需要经过评估、选择、交叉、变异四个操作，产生新一代种群。

1. **评估**。要使种群趋向更优，首先需要了解每个个体的 “适应度”。此步骤对种群的所有个体进行适应度评估，通过适应度函数计算出当前种群的适应度向量，并记录当前最大适应度和最优个体。
2. **选择**。以轮盘赌算法为例（其余选择算法将在拓展研究中讨论），每个个体 x_i 被选入新种群的概率 $p(x_i)$ 等于其适应度 $f(x_i)$ 在种群所有个体适应度之和中的占比，即

$$p(x_i) = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)} \tag{1}$$

   式中，N 表示种群规模。共进行 N 次选择，即可生成一个新的种群，其适应度期望应当优于原种群。
3. **交叉**。经过选择，种群中会存在很多相同的个体，此时为了个体的多样性，需要进行交叉和变异操作。本例中采取两点交叉的算法，随机生成两个介于 1 至 90 之间的整数 l 和 r，对两个个体 x_i 和 x_j 的 l 至 r 基因片段进行交换，如图1。本例中，采取对相邻个体进行交叉的策略，即 $j = i+1$。

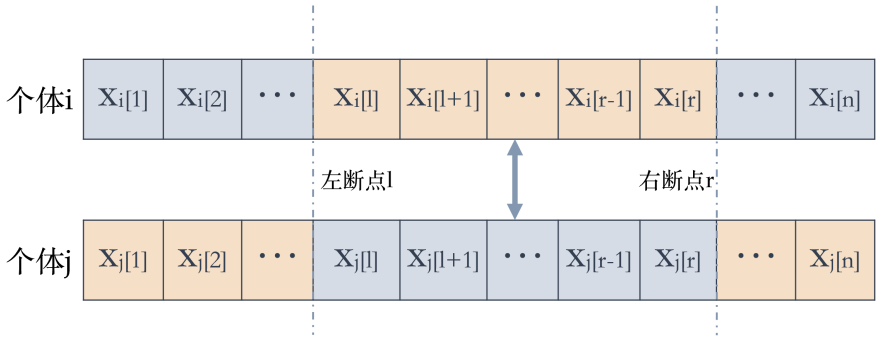

图 1　两点交叉算法示意

4. **变异**。在较为成熟的种群中，会有大量相同的个体，此时交叉不能提供任何多样性。设置一定的变异概率 p_v，对任意个体的任意基因，都有一定概率取反。在基础的算法中，p_v 是一个定值，但对变异概率的调整优化在本例中对算法收敛速度和“早熟”情况的避免都起着至关重要的作用，后文中将对其进行讨论。

设置迭代次数或终止条件（如 k 代以内没有出现更优个体），即可通过遗传算法计算出大部分优化类问题的较优解。

4.1.2 适应度计算

本例中的适应度在要求中给出，计算方法可分为以下几步：

1. 根据个体 i 选定的标定温度及其对应的电压，对训练集中所有条目，通过三次样条插值 [1]，预测所有待测电压对应的温度，记为 $\hat{T_{i,1}}, \hat{T_{i,2}}, \cdots, \hat{T_{i,n}}$。
2. 计算总成本 C_{total}，其值由2式给出。

$$C_{total} = QkN + \sum_{i=1}^{N}\sum_{j=1}^{90} s_{i,j} \tag{2}$$

式中，Q=80，表示一次标定的成本；k 表示个体中 true 的数量，即每次完整标定的次数；$s_{i,j}$ 由3式给出。

$$s_{i,j} = \begin{cases} 0 & \left|\hat{T_{i,j}} - T_{i,j}\right| \le 0.5 \\ 1 & 0.5 < \left|\hat{T_{i,j}} - T_{i,j}\right| \le 1.0 \\ 10 & 1.0 < \left|\hat{T_{i,j}} - T_{i,j}\right| \le 1.5 \\ 30 & 1.5 < \left|\hat{T_{i,j}} - T_{i,j}\right| \le 2.0 \\ 80000 & \left|\hat{T_{i,j}} - T_{i,j}\right| > 2.0 \end{cases} \tag{3}$$

3. 计算适应度 $f(x_i)$，为了进一步拉开优质个体与其他个体之间的差距、加速收敛，采取对平均成本统一减去 400 后再取倒数的策略，即

$$f(x_i) = \frac{1}{\frac{C_{total}}{N} - 400} \tag{4}$$

4.2 改进的双种群遗传算法

双种群遗传算法 (Dual Population Genetic Algorithm, DPGA)[2] 是对简单遗传算法的一种优化。该算法设置了两个相对独立而又互相交流的种群，能更好地保持多样性、一定程度上避免早熟，也能通过最优个体的交流加快收敛。算法的流程如图2，对两个种群分别初始化、评估、选择、交叉、变异，在此基础上增加“移民”操作。

移民，即间隔一定代数（本例中取 5 代），用一个种群的最优个体替代另一个种群的最劣个体。常规的双种群遗传算法一般采用“淘汰种群”的概念，仅将该种群的最优个体转移到常规种群中；而针对此优化问题，采取改进的双种群遗传算法，设置两个平等的种群，互相用最优个体替代对方的最劣个体，更好地促进种群之间的交流，加速优化进程。

改进的双种群遗传算法伪代码见算法1。

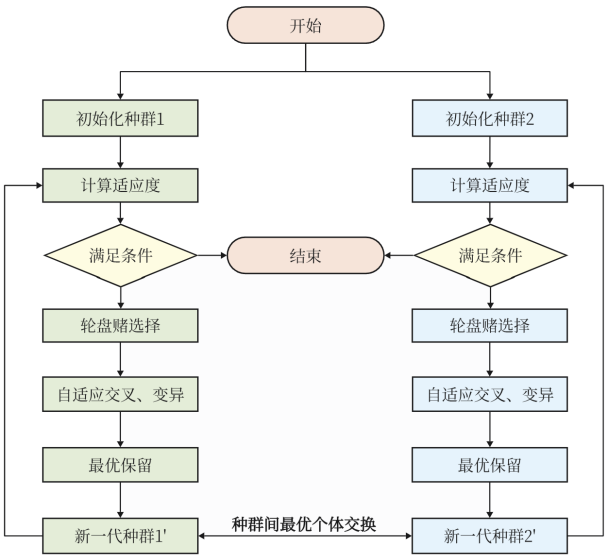

图 2　双种群遗传算法示意

算法 1 改进的双种群遗传算法

输入: 适应度函数 F，选择函数 S，代数 G，种群大小 N，概率参数向量 P

$X, Y \leftarrow randomized\ matrix$
for $g = 1 \rightarrow G$ **do**
    $fitval1, fitval2 \leftarrow F(X), F(Y)$
    $newX, newY \leftarrow S(fitval1, X), S(fitval2, Y)$
    for $i = 1 \rightarrow N\ Step\ 2$ **do**
        if $rand() < P(crossover)$ **then**
            $l, r \leftarrow randi(1, 90)$
            $newX(i, l:r) \leftrightarrow newX(i+1, l:r)$
            $newY(i, l:r) \leftrightarrow newY(i+1, l:r)$
        end if
    end for
    for $(i, j) = (1, 1) \rightarrow (N, 90)$ **do**
        if $rand() < P(mutation)$ **then**
            $newX(i, j) \leftarrow \neg newX(i, j)$
            $newY(i, j) \leftarrow \neg newY(i, j)$
        end if
    end for
    $newX(1), newY(1) \leftarrow best\ individual\ of\ X, Y$
    if $g \equiv 0 (mod\ 5)$ **then**
        $newX \leftarrow best\ individual\ of\ Y$
        $newY \leftarrow best\ individual\ of\ X$
    end if
end for

4.3 参数的确定和优化

4.3.1 种群规模

本例中，单个个体由 90 位 0/1 表示，为了确保多样性，避免陷入局部最优，应当设置较大的种群规模 N（N 代表单个种群的个体数量，即实际算法中总个体数为 2N）。考虑到过大的 N 会导致计算量增加、运行效率降低，故设计实验确定一个较佳的种群规模，测试 50 代以内某规模所得到解与最优解的接近程度、时间成本和稳定性。测试平台为 AMD Ryzen 6800U（电池模式），Matlab R2023b。实验重复五次，表1中的数据系五次实验的统计值。

表 1　种群规模确定实验数据

种群规模 N	平均完成时间 (s)	最佳成本	成本总体标准差 σ
100	211	468.86	7.4
150	309	466.37	5.6
200	407	458.97	2.8
250	503	458.97	2.2

结合实验结果，综合考虑多种因素，最终确定选用 200 作为单个种群的规模。此时，设置代数为 150 时，进行的 15 轮测试中有 12 次收敛到 458.97。

4.3.2 自适应交叉率和变异率

为了进一步避免早熟情况，可以使交叉率和变异率根据当前个体在种群中的情况自适应变化 [3]。为了让处于劣势的个体尽早淘汰、变异，同时减少对优势个体的基因破坏，可以如下设置交叉或变异率。

$$P=\begin{cases} P_1-\frac{(P_1-P_2)(f(x_i)-\bar{f})}{f_{max}-\bar{f}},\ f(x_i)\geq\bar{f} \\ P_1,\ f(x_i)<\bar{f} \end{cases} \tag{5}$$

式中，$f(x_i)$ 为待交叉个体的较小适应度值或待变异个体的适应度值，$\bar{f}$ 为种群平均适应度，f_{max} 为种群最大适应度。在本例中，由于问题的搜索空间多达 2^{90} 种，故交叉概率可以设置得较大，从而在一定时间内搜索更多解，因此对于交叉操作，$P_1=1, P_2=0.8$；而变异概率则不能过大（4.3.3中将进一步讨论），设置 $P_1=0.05, P_2=0.02$。

4.3.3 基于 Bayes 公式干预的变异概率控制

对于变异概率，一般对不同的基因片段内容使用相同的规则。但在本题中，个体中更多的 true 会导致更高的成本（更低的适应度），且实际上最优解中 false 的数量是远远多余 true 的。如果对 true 和 false 使用相同的变异概率，假设某个经过选择的种群中 true 的数量为 G_T，false 的数量为 G_F，变异概率为 p，则变异后种群中 true 数量的期望 $E(G'_T)$ 由6式给出。

$$E(G'_T) = G_F \cdot p + G_T \cdot (1-p) = p(G_F - G_T) + G_T \tag{6}$$

这个期望对变异前的 G_T 作商可得7式。

$$\frac{E(G'_T)}{G_T} = p(\frac{G_F}{G_T} - 1) + 1 \tag{7}$$

显然，$G_F > G_T$，且 $G_F/G_T \gg 1$，因此变异会导致种群中 true 的数量增多。在此基础上，如果没有选择操作，则种群最终必然趋向 true 和 false 数量的平衡。虽然算法中的选择操作会淘汰掉其中的劣势个体，但为了能让淘汰效率对抗变异导致的比例失调，就需要调低变异概率，造成迭代后期多样性不足。

为了解决这个问题，本例采取一种基于 Bayes 公式干预的变异概率控制。该算法的目标就是使变异过程产生更多趋近最优解的个体，保证种群中 true 和 false 数量的合理性。对当前得到的最优个体 $X(i)$，记 $G_F(i) = g_f$, $G_T(i) = g_t$，如下设置种群中 true 和 false 分别变异的概率系数 ρ_t 和 ρ_f。

$$\rho_t = \frac{g_f}{g_f + g_t}, \quad \rho_f = \frac{g_t}{g_f + g_t} \tag{8}$$

ρ_t 和 ρ_f 是关于代数 gen 的函数。综合5式和8式，可以得到第 gen 代第 i 个个体、第 j 个基因片段的变异概率由9式给出。

$$p_v(gen, i, j) = \rho \cdot \frac{P(g_t + g_f)}{2g_t} \tag{9}$$

4.4 几种运算效率优化方案

为了在算法确定的情况下提高代码的运行速度，本文给出了如下四种针对性优化方案：

1. 初始化种群时，使 false 的数量大于 true，提供尽可能更好的初始种群。具体实现上，可以对随机数进行乘方运算后再取整。
2. 使用 logical 形式的数组存储种群。由于本例中基因仅由 0 和 1 构成，故使用 logical 数据类型能够在尽可能少占用资源的情况下正常实现功能。
3. 使用 persistent 变量存储训练数据。如果每次调用适应度函数都读取一次训练集，会消耗较多时间；global 变量则违背模块化编程的思想。使用 persistent 变量可以避免以上问题。
4. 在进行任何大型矩阵的赋值前，先将矩阵设置为目标大小的零矩阵。在 MATLAB 中，修改数据比逐个添加数据效率要高得多。

4.5 结果展示

经过多轮运行，得到的最优解为

$$[5 \quad 22 \quad 45 \quad 73 \quad 88]$$

其对应的测试集平均成本为 458.9685。

对于此最优解和过程中得到的两个次优解进行误差成本稳定性分析，得到图3。其中，次优解 1 为 [4 22 45 73 88]，其平均成本为 463.52；次优解 2 为 [4 23 46 74 87]，其平均成本为 466.37。分析图像可知，所得的最优解不仅平均成本最低，且误差成本最稳定、高误差出现频率最小、峰值最低。

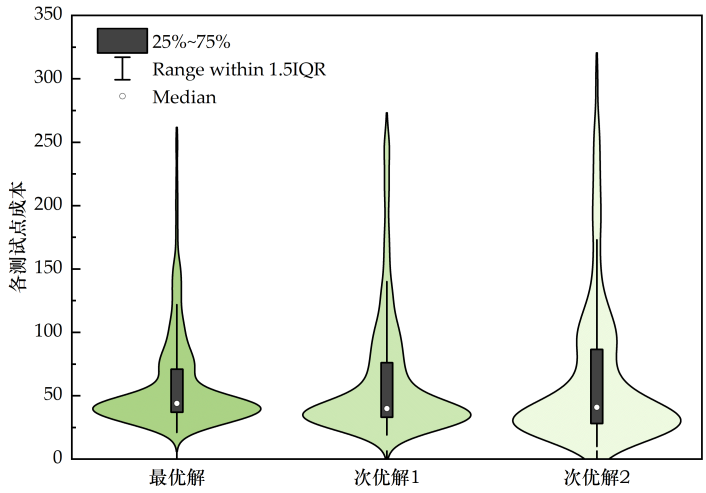

图 3　三个较优解在测试集中的误差成本及其稳定性

其中一次收敛至最优解的收敛过程如图4。在这次收敛过程中，从代数 79 开始，最优解从 6 个测试点变为 5 个测试点，而这也是平均成本由逐渐变小转变为逐渐增大的分界线。其原因可能和5.1中最优解在测试集中表现不佳的原因类似，即 5 个测试点的个体很容易在某些温度上存在较大误差，而随着 5 个测试点的精英个体出现，配合4.3.3中提出的变异概率计算方法，种群中 5 个测试点的个体会逐渐变多，而出现的较大误差点也会增加，导致平均成本不降反增。

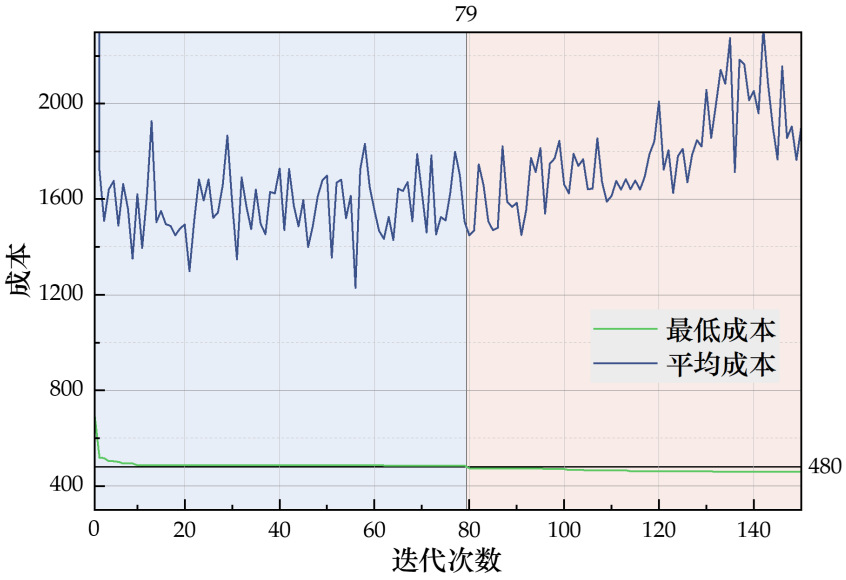

图 4　一次收敛至最优解的过程

五、 模型测试与评价

5.1 在测试集中的表现

使用测试集 A 和 B 对上述结果进行测试，所得的平均成本分别为 619.62 和 619.45。这个结果远高于训练集得到的 458.97。对每个测试点的成本进行分析，发现在两个测试集中各有一个测试点出

现了超过 2°C 的误差，导致了很大的惩罚值。对其他次优解进行测试，所得平均成本情况如表2。

表 2　种群规模确定实验数据

次优解	训练集平均成本	测试集 A 平均成本	测试集 B 平均成本
[4 22 45 73 88]	463.52	464.64	464.36
[4 23 46 74 87]	466.37	468.86	466.46
[3 12 26 54 73 88]	488.67	488.54	489.36
[2 15 24 52 75 87]	485.12	485.21	485.05

从表中可以看出，几个次优解在测试集中的表现都相对稳定。因此，最优解可能存在广义的“过拟合”问题，如果要减小误差，可以增加测试集规模，也可以考虑放弃一定的成本，选用六个测试点的方案进行标定。

5.2 与简单遗传算法的比较

为了测试改进的双种群遗传算法相较于简单遗传算法是否确实进步，设置简单遗传算法的种群规模 $N' = 2N$，以抹平双种群带来的个体数量差距，再进行收敛效率、结果稳定性两方面的测试。测试平台为 Intel Core i7-12700H，Matlab R2023b。

1. **收敛效率**。对两种算法分别进行 5 次测试，每次进行 150 代，迭代过程中的当前最低成本曲线如图5。其中，每代对应的数据为 10 次测试的均值。虽然简单遗传算法在初期更快得到了一批 485 左右的较优解，但改进的双种群遗传算法更快得到了五个标定点的解，且得到最优解的次数也明显更多。

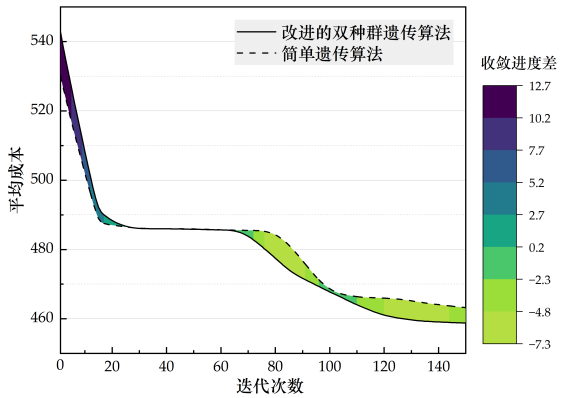

图 5　两种算法收敛效率和效果对比

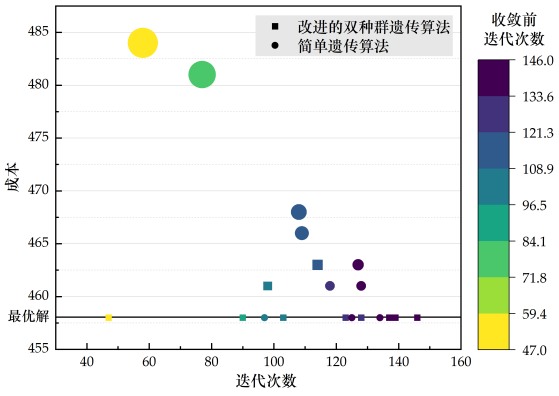

图 6　两种算法收敛结果稳定性对比

2. **结果稳定性**。设置算法终止条件为连续 80 代不再出现更优解，分别进行 10 次实验，得到两种算法最终得到的成本和收敛起始点的散点图如图6。显然，改进后的双种群遗传算法具有更强的稳定性，早熟现象出现较少，收敛效率也略优于简单遗传算法。

5.3 模型评价与改进方向

结合上述测试，总结模型的优势如下：

1. 算法的收敛效率明显胜过简单遗传算法，且早熟情况明显减少，适合用于进行类似优化问题的训练；
2. 提出了一种改进的双种群遗传算法，实现方式简单且效果尚佳；
3. 运用自适应交叉率和变异率，且针对问题创新地给出了基于 Bayes 公式干预的变异概率控制。

此外，该模型仍然存在一些有待改进之处，如纯粹的基于平均成本的单目标优化会导致所得答案普适性不高，对测试集效果远不及训练集，存在广义的“过拟合”情况；每次运算时仍有一定概率出现早熟的情况。对此，可以将问题转化为多目标优化问题，采用层次分析法，并设置动态权重以提高结果的普适性；可以考虑其他优化算法，如模拟退火算法、粒子群算法等，一定程度上避免早熟问题的出现。

六、 拓展研究：关于“选择”的算法优劣

除了交叉、变异概率和种群规模等参数之外，选择算法的选取同样对整体算法有至关重要的影响。对当前优秀个体的过度选择容易陷入局部最优，而一味维持个体多样性则会拖慢收敛速度。为了解针对本问题的较优选择算法，进行以下研究。轮盘赌算法在前文中已经经过讨论，故在此不作介绍。

6.1 待研究的选择算法

6.1.1 锦标赛选择 (Tournament Selection)

该算法进行若干次“锦标赛”（本题中次数即种群规模），每次有放回地从种群中随机取出 k 个个体，并将其中的最佳个体放入新种群。为了避免过高的重复度，往往设置较小的 k（如 2,3）。相对于以轮盘赌算法为代表的适应度值比例选择算法，锦标赛选择算法噪声较小，最优个体在其参与的所有竞赛中都会获胜，而最劣个体永远不会获胜。算法示意如图7。

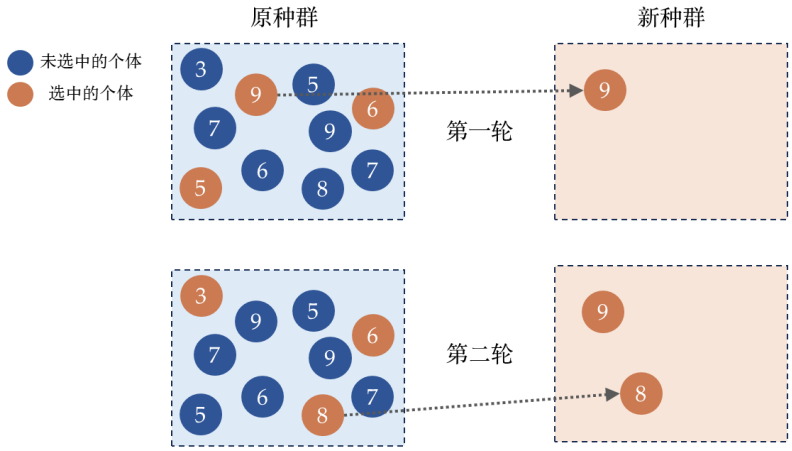

图 7　k=3 的锦标赛选择前两轮

6.1.2 截断选择 (Truncation Selection)

该算法较为朴素，直接在当前种群中重复选取适应度值为前 k 的个体，直至达到需要的新种群规模。该算法噪声为零，理论收敛速度最快，而对于较小的种群，容易导致多样性极小。但本题中，需要的种群规模较大（见4.3.1），故可以选取较多个体，实际操作中不一定显著弱于其他算法。

6.1.3 随机遍历选择 (Stochastic Universal Sampling, SUS)

该算法是对轮盘赌算法的一个改变。为了丰富多样性，进一步提升不公平性以保留存在潜力的劣势个体，此算法在累计适应度值的基础上等间距取值。具体实现方法为：

1. 计算取值间距 $p = \sum_{i=1}^{N} f(x_i)/N$
2. 在 $[0, p)$ 范围内随机取值作为起始指针 p_0
3. 计算所有指针的位置，即 $p_0, p_0 + p, p_0 + 2p, \cdots, p_0 + (N-1)p$
4. 根据指针位置在累计适应度中找到对应的个体，放入新种群

这种算法相当于对多次随机过程进行了规范，理论上是对轮盘赌算法的一种优化。

6.2 不同算法的比较实验

虽然不同的算法在时间复杂度上有显著区别，但由于本问题的主要时间消耗在于三次样条插值而非遗传算法中的主干操作，故此处可以忽略。因此，仅针对本题，对上述算法的收敛效率、结果稳定性进行实验。

为了更好地评价不同选择算法的收敛效率、结果稳定性，需要对两者进行规范化。收敛效率一般由达到某一成本值以下所需代数评判，而结果稳定性一般考虑收敛到最优解的频率。综合考虑之前的实验中得到的经验，定义收敛效率 ν 和结果稳定性 ς 如下：

- 收敛效率 ν 为 10 次实验的平均历史成本函数首次低于 480 所对应的迭代次数的倒数。
- 结果稳定性 ς 为 10 次实验中收敛到 458.9685 次数占总实验次数的比例。

采用上述改进的双种群遗传算法，设置单个种群规模为 200，代数为 150，对四种算法分别进行 10 次测试。测试平台为 Intel Core i7-12700H, Matlab R2023b 和 Intel Core i9-14900K, Matlab R2023b。测试结果如图8，各算法的 ν, ς 值及其排名如表3。

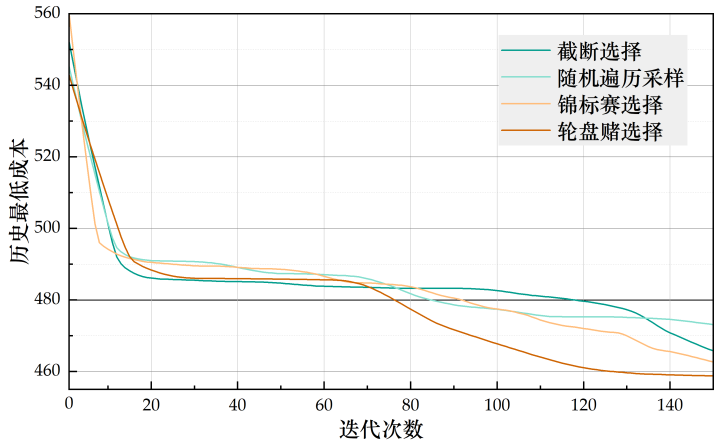

图 8　四种选择算法收敛情况对比

表 3　四种选择算法收敛效率及结果稳定性

选择算法	收敛效率 ν	结果稳定性 ς	收敛效率排名	结果稳定性排名
轮盘赌选择	13.0×10^{-3}	0.7	1	1
锦标赛选择	10.9×10^{-3}	0.5	3	2
截断选择	8.40×10^{-3}	0.5	4	2
随机遍历选择	11.8×10^{-3}	0.3	2	4

6.3 实验结果分析

本次实验与理论分析结果有所出入。轮盘赌选择在本题中仍然是四者中最优；随机遍历选择理论上优于轮盘赌算法，却在实际情况下结果稳定性较差；截断选择十分朴素，选择方法激进，却在收敛效率上表现不佳，反而在结果稳定性上表现尚可。这些情况的原因分析如下：

1. 受限于算力和时间限制，未能进行足够次数的实验，结果仍然存在一定的偶然性；
2. 遗传算法本身是一种以随机为根本的算法，每次实验之间都存在较大的数据差别，个别组的严重早熟状况会导致多组平均值的较大偏差，使得数据不能完全反映算法的表现；
3. 每种算法本身没有绝对的优劣之分，在本题中表现的排名不能代表所有情况，使用改进的双种群遗传算法和使用简单遗传算法也可能得到不同的实验结果；
4. 前述所有优化都是在轮盘赌选择算法的背景下进行的，如果更换选择算法，可能需要相应地更改其他参数。遗传算法的不同环节看似分立，实则环环相扣。

依然可以确定的是，轮盘赌选择是针对解决本问题较好的一种选择算法；其他选择算法虽然表现不及轮盘赌算法，但同样可以通过重复试验得到相同的最优解。

参考文献

[1] 许小勇, 钟太勇. 三次样条插值函数的构造与 Matlab 实现[J]. 兵工自动化,2006,(11):76-78.

[2] 欧阳飞羽, 王巍. 基于改进双种群遗传算法的装配线平衡优化研究[J]. 中国新技术新产品,2023(23):118-121. DOI:10.13612/j.cnki.cntp.2023.23.030.

[3] 黄哲, 颜廷旭. 基于自适应遗传算法的柔性机械手抑振轨迹优化[J]. 工业控制计算机,2023,36(11):121-123.

[4] 熊邦书, 黄武涛, 李新民. 基于改进遗传算法的摄像机标定参数优化方法[J]. 半导体光电,2016,37(01):110-114+118. DOI:10.16818/j.issn1001-5868.2016.01.024.

附录 A　核心代码片段

1.1　改进的双种群遗传算法主程序

```
clear;
n=90;
pop_size=200;
gens=150;

%初始化
pop1=round(rand(pop_size,n).^15);
pop2=round(rand(pop_size,n).^15);
pop1=logical(pop1);              % logical减小内存占用，可能提升效率
pop2=logical(pop2);
%适应度记录
prev_best_fit=zeros(gens,1);
prev_avg_cost=zeros(gens,1);

for gen=1:gens
    disp(['正在进行第',num2str(gen),'次迭代']);
    %评估
    fit_val1=arrayfun(@(index) fitness(pop1(index,:)),1:pop_size)';
    [best_fit1,bestIdx1]=max(fit_val1);
    best_indiv1=pop1(bestIdx1,:);
    fit_val2=arrayfun(@(index) fitness(pop2(index,:)),1:pop_size)';
    [best_fit2,bestIdx2]=max(fit_val2);
    best_indiv2=pop2(bestIdx2,:);

    %记录历史
    prev_best_fit(gen)=max(best_fit1,best_fit2);
    disp([1/best_fit1+400,1/best_fit2+400]);



    newpop1=false(size(pop1));
    newpop2=false(size(pop2));
    %选择
    %轮盘赌
    cumul_fit=cumsum(fit_val1)/sum(fit_val1);
    for i=1:pop_size
        selected=rand();
        selected_index=find(cumul_fit>=selected,1);
        newpop1(i,:)=pop1(selected_index,:);
    end
    cumul_fit=cumsum(fit_val2)/sum(fit_val2);
    for i=1:pop_size
        selected=rand();
        selected_index=find(cumul_fit>=selected,1);
```

```
    newpop2(i,:)=pop2(selected_index,:);
end
%锦标赛
% selected_index1=TournamentSelection(fit_val1,3,pop_size);
% selected_index2=TournamentSelection(fit_val2,3,pop_size);
%截断选择
% selected_index1=TruncationSelection(fit_val1,15,pop_size);
% selected_index2=TruncationSelection(fit_val2,15,pop_size);
%随机遍历选择
% selected_index1=StochasticUniversalSampling(fit_val1,pop_size);
% selected_index2=StochasticUniversalSampling(fit_val2,pop_size);
% for i=1:pop_size
%    newpop1(i,:)=pop1(selected_index1(i),:);
% end
% for i=1:pop_size
%    newpop2(i,:)=pop2(selected_index2(i),:);
% end

%交叉
f_avg1=mean(fit_val1);
f_avg2=mean(fit_val2);
prev_avg_cost(gen)=2/(f_avg2+f_avg1)+400;
disp(prev_avg_cost(gen));
for i=1:2:pop_size
    if min(fit_val1(i),fit_val1(i+1))<f_avg1
        cross_rate=1;
    else
        cross_rate=1-0.2*(fit_val1(i)-f_avg1)/(best_fit1-f_avg1);
    end
    if rand()<cross_rate
        t1=randi(n);t2=randi(n);
        cross_pnt_left=min(t1,t2);
        cross_pnt_right=max(t1,t2);
        newpop1(i,cross_pnt_left:cross_pnt_right)=newpop1(i+1,cross_pnt_left:cross_pnt_right);
        newpop1(i+1,cross_pnt_left:cross_pnt_right)=newpop1(i,cross_pnt_left:cross_pnt_right);
    end
    if min(fit_val2(i),fit_val2(i+1))<f_avg2
        cross_rate=1;
    else
        cross_rate=1-0.2*(fit_val2(i)-f_avg2)/(best_fit2-f_avg2);
    end
    if rand()<cross_rate
        t1=randi(n);t2=randi(n);
        cross_pnt_left=min(t1,t2);
        cross_pnt_right=max(t1,t2);
        newpop2(i,cross_pnt_left:cross_pnt_right)=newpop2(i+1,cross_pnt_left:cross_pnt_right);
        newpop2(i+1,cross_pnt_left:cross_pnt_right)=newpop2(i,cross_pnt_left:cross_pnt_right);
```

```
    end
end

%变异
if gen==1
    mut_rate_ones=0.92;
    mut_rate_zeros=0.08;
else
    mut_rate_zeros=sum(best_indiv1)/n;
    mut_rate_ones=1-mut_rate_zeros;
end
for i=1:pop_size
    if fit_val1(i)<f_avg1
        mut_rate=0.05*n/2/sum(best_indiv1);
    else
        mut_rate=(0.05-0.03*(fit_val1(i)-f_avg1)/(best_fit1-f_avg1))*n/2/sum(best_indiv1);
    end
    for j=1:n
        if newpop1(i,j)==true && rand()<mut_rate*mut_rate_ones || newpop1(i,j)==false &&
            rand()<mut_rate*mut_rate_zeros
           newpop1(i,j)=~newpop1(i,j);
        end
    end
end
newpop1(1,:)=best_indiv1;

if gen==1
    mut_rate_ones=0.92;
    mut_rate_zeros=0.08;
else
    mut_rate_zeros=sum(best_indiv2)/n;
    mut_rate_ones=1-mut_rate_zeros;
end
for i=1:pop_size
    if fit_val2(i)<f_avg2
        mut_rate=0.05*n/2/sum(best_indiv2);
    else
        mut_rate=(0.05-0.03*(fit_val2(i)-f_avg2)/(best_fit2-f_avg2))*n/2/sum(best_indiv2);
    end
    for j=1:n
        if newpop2(i,j)==true && rand()<mut_rate*mut_rate_ones || newpop2(i,j)==false &&
            rand()<mut_rate*mut_rate_zeros
           newpop2(i,j)=~newpop2(i,j);
        end
    end
end
newpop2(1,:)=best_indiv2;
```

```
    %移民
    if rem(gen,5)==0
        newpop2(find(fit_val2==min(fit_val2),1),:)=best_indiv1;
        newpop1(find(fit_val1==min(fit_val1),1),:)=best_indiv2;
    end

    %table=array2table(newpop);
    %writetable(table,'newpop.xlsx');
    %table=array2table(pop);
    %writematrix(pop,'pop.xlsx');
    %更新种群
    pop1=newpop1;
    pop2=newpop2;
end

disp('最优选择：');
if best_fit1>best_fit2
    best_indiv=best_indiv1;
    cost=1/best_fit1+400;
else
    best_indiv=best_indiv2;
    cost=1/best_fit2+400;
end
disp(find(best_indiv));
disp(['最低成本：' num2str(cost)]);
prev_low_cost=1./prev_best_fit+400;
%table=array2table(prev_low_cost);
%writetable(table,'选择-SUS.xlsx');
x=1:gens;
plot(x,prev_low_cost,'-*b',x,prev_avg_cost,'-or');
axis([0,160,400,2400])
set(gca,'XTick',0:20:160)
set(gca,'YTick',400:400:2400)
legend('最低成本','平均成本');
xlabel('x')
ylabel('y')
```

1.2 适应度计算函数

```
function fitness=fitness(individual)
    % 只读取一次数据提高效率
    persistent data;
    if isempty(data)
        table=readtable("dataform_train2024.csv");
```

```
        data=table2array(table);
    end

    % 三次样条插值、多项式拟合
    datanum=2000;
    X=find(individual);
    len=length(X);
    if len<2
        fitness=0;
        return;
    end
    yi=data(2:2:2*datanum,1:90);
    y=data(2:2:2*datanum,X);
    V=zeros(datanum,90);
    for i=1:datanum
        V(i,:)=interp1(y(i,:),X,yi(i,:),'spline');
    end
    %table=array2table(Y);
    %writetable(table,"Y.csv");

    % 成本计算
    fitness=80*length(X);
    delta=abs(V-(1:90));
    %disp(delta);
    s1=sum(sum(delta>0.5));
    s2=sum(sum(delta>1))*9;
    s3=sum(sum(delta>1.5))*20;
    s4=sum(sum(delta>2))*79970;
    fitness=fitness+(s1+s2+s3+s4)/datanum;
    fitness=1/(fitness-400); % -400，突出优质个体的适应度优势，提高收敛速度

end
```

1.3 锦标赛选择、截断选择、随机遍历选择算子

```
% 锦标赛选择
function [selected_indices] = TournamentSelection(fitness, tournamentSize, numSelected)
    populationSize = length(fitness);
    selected_indices = zeros(1, numSelected);

    for i = 1:numSelected
        tournamentContestants = randperm(populationSize, tournamentSize);
        [~, bestIndex] = max(fitness(tournamentContestants));
        winner = tournamentContestants(bestIndex);
        selected_indices(i) = winner;
```

```
    end
end
%截断选择
function [selected_indices] = TruncationSelection(fitness, selection, pop_size)
    sorted=sort(fitness);
    selected_indices=zeros(1,pop_size);
    for i=1:pop_size
        selected_indices(i)=find(fitness==sorted(pop_size-rem(i-1,selection)),1);
    end
end
%随机遍历选择
function selected = StochasticUniversalSampling(fitness,pop_size)
    [ind,~] = size(fitness);
    cumfit = cumsum(fitness);
    trials = cumfit(ind) / pop_size * (rand + (0:pop_size-1)');
    f = cumfit(:, ones(1, pop_size));
    t = trials(:, ones(1, ind))';
    [selected, ~] = find(t < f & [ zeros(1, pop_size); f(1:ind-1, :) ] <= t);
    [~, shuf] = sort(rand(pop_size, 1));
    selected = selected(shuf);
end
```