Robust Optimization Over Time by Estimating Robustness of Promising Regions

摘要

许多真实世界的优化问题是动态的。随着时间的推移鲁棒优化（ROOT）领域处理动态优化问题，其中不希望频繁更改已部署的解决方案。原因是受制于切换部署解决方案的高昂代价，部署一个新解决方案的所需的资源的限制，以及系统对频繁变化的低容忍性。本文所考虑的 ROOT 问题的主要目标是找到在环境变化中能坚持最久的解决方案。在为解决这些问题而开发的最先进的方法中，用于选择部署解决方案的决策者/指标大多对问题实例做出简化的假设。此外，当前的方法都使用种群控制组件，这些组件最初是为了随着时间的推移跟踪全局最优解而设计的，而没有针对任何鲁棒性考虑。本文提出了一个多种群 ROOT 算法包含两个新颖的组件：1）一个估计可行域鲁棒性的鲁棒估计组件；2）一个用于管理子种群的双模式计算资源分配组件。实验结果证明了本文提出的算法相对于目前最先进算法的优越性。

关键词：计算资源分配（CRA）；进化动态优化（EDO）；多种群；随时间推移的鲁棒优化（ROOT）；鲁棒估计

1 引言

本论文聚焦于 ROOT 问题，即 Robust Optimization Over Time 问题。涉及动态优化领域中的一个重要挑战：在经常变化的环境中找到稳健的解决方案。这类问题特别关注那些由于高转换成本、资源限制或系统不适合频繁更改而无法频繁更换解决方案的场景。ROOT 问题的核心在于如何在持续变化的条件下，找到不仅当前有效但对未来变化也有一定抵抗力（高鲁棒性）的解决方案。这种优化问题在许多实际应用中非常重要，比如能源管理、供应链优化和自适应控制系统等领域。有效的 ROOT 策略可以提高决策过程的稳定性和效率，减少由于环境变化而导致的成本和风险。通过研究 ROOT 问题，可以开发出更加智能、适应性强的算法和系统，以更好地应对现实世界的复杂性和不确定性。

2 相关工作

由于 ROOT 问题需要应对动态变化的环境，所以解决 ROOT 问题的算法通常由进化动态优化算法与一些其他额外组件构成。接下来将从 ROOT 问题的定义与 ROOT 算法研究进展来个方面来介绍相关工作。

2.1 ROOT 问题定义

通常可将 ROOT 问题分为三类：1）用一个接受度指标来决定在一个环境中是否接受部署一个解，并寻求部署持续时间最久的解 [9]。在这类问题中，已部署的解决方案将在连续的环境中重复使用，直到其质量在新环境中下降到可接受的水平以下。在实际应用中，此类问题是指已部署解决方案的“频繁更改”违反了系统要求，导致系统不稳定或例如例如安全性、客户满意度等重要因素恶化。在这些情况下，只要解仍然可以接受就将其保留。部署解可接受的更改频率具体取决于问题和应用程序的具体要求，例如适应新部署的解所需的时间、系统的稳定性、部署新解的成本。解决方案。2）拥有时间窗的 ROOT 问题，其中每个时间窗包含数个环境，但只能在每个时间窗开始时选择一个解部署，并争取该解在本个时间窗中一直保持有效 [1]。与第一类问题不同，此类问题中没有决定解的接受程度的阈值，其目标是最大化每个时间窗口上适应度值的平均值。3）平衡已部署解的可接受性和切换成本的 ROOT 问题，其中已部署的解将被一直保留，直到它变得不可接受或发现另一个解决方案其适用性远高于已部署的解决方案，从而使切换的带来的收益超过切换的代价 [4]。本篇论文针对上述第一类问题，其优化目标可以描述为

$$\min\{l|1 \leq l \leq T\}$$

其中 l 表示部署解的数量，T 表示总的环境数。同时如果一个解能持续在两个及以上的环境中保持可接受，则该解被称为鲁棒解。每个环境对应一个相应的目标函数 $f^t(\vec{x})$，一个解的目标函数值被称为适应度，若解的适应度大于等于预定义的质量门槛 μ，则称该解为可接受的解。若之前环境中部署的解在当前环境下不满足质量门槛，则需要选择一个新的解部署。

2.2 ROOT 算法研究进展

目前为止 ROOT 领域的研究工作还不是很充分，大多数工作都是在三个主要的 ROOT 算法框架的基础上开展。金耀初等人 [2]，提出通过预测解在一定数量的未来环境中的适应度值来评估其鲁棒性。该方法的主要组件包括单种群的进化动态优化（EDO）算法、数据库、近似器（也称代理模型）和预测器。单种群 EDO 负责收集数据以训练预测器，并进行优化过程。数据库用于存储历史数据，并将其用于训练近似器和预测器。该方法的目标函数为实际适应度、预测值和近似值的累积。Haobo Fu 等人 [1]，提出了一种针对基于生存时间的 ROOT 算法，该算法的主要组件与金耀初等人的方法类似，不同之处在于其目标函数为最大化解的生存时间，即使解能经受尽可能多的环境变化。Yazdani 等人 [5]，提出一种基于可靠性 ROOT 的方法，该方法直接在原始目标函数的搜索空间中进行搜索，其主要包括一个多种群的 EDO 组件，负责定位和追踪多个移动的可行区域，然后再通过一个决策组件选择一个解来进行部署。

文章指出现有 ROOT 算法大多数是由动态优化领域的算法进行简单的改变后套用过来，缺乏专门为 ROOT 问题设计的针对性框架。同时，目前许多 ROOT 领域的算法是基于预测候选解未来的适应度而估计其鲁棒性，这其中需要使用多个过去环境的拟合器与训练预测器，这种方法存在很大的误差。基于以上两点不足，本文提出了一个基于可行域鲁棒性估计的针对性 ROOT 算法框架。

3 本文方法

3.1 本文方法概述

算法的主体框架包括动态优化算法的几个经典部分，例如多种群机制、多样性控制机制、变化响应机制。在此基础上加入了针对 ROOT 问题开发的新机制，计算资源分配机制（CRA）、鲁棒性估计机制（RE）。其中动态优化算法负责在变化的环境中追踪与定位各环境下的有希望区域，并在环境发生变化时做出对变化的响应。基于系统的双模式鲁棒性的 CRA 组件，在每次迭代开始时选择运行哪些子种群，以降低用于评估适应度的计算开销。该机制的决定因素包括系统状态、每个有希望区域的鲁棒性以及子种群的角色和任务完成度。鲁棒性估计组件 RE，使用显式存档来保持和迁移关于有希望区域的历史知识，用于估计其鲁棒性。有希望的区域的鲁棒性值用于 CRA 选择子种群与部署下一个解。

3.2 动态优化算法模块

在 ROOT 中，多种群 EDO 的主要职责不是像传统 EDO 一样寻找全局最优，而是定位和跟踪多个移动的有希望的区域。每个子种群跟踪并覆盖一个有希望的区域。其主要目的是，在每次环境变化之后，有希望区域的峰顶周围的解决方案更有可能依然满足质量门槛。使用一个鲁棒性估计组件的显式档案，检索先前环境中每个子种群的最佳位置的历史信息，并用于估计每个覆盖区域的鲁棒性。

本文所提出的方法，可以兼容各种 EDO 算法。此次复现工作，依据论文作者的建议使用一个多种群的粒子群优化算法（PSO）作为动态优化组件。该 EDO 的主要流程为：1）首先初始化一个子种群；2）每次迭代依次运行每个子种群，按 PSO 算法进行搜索；3）同时计算每个没饿过子种群的覆盖区域的空间大小，若其空间尺寸小于一定阈值则说明该子种群已收敛，则停止该子种群的优化；4）当环境发生变化时，将停滞的子种群重新随机初始化，已保持追踪变化的环境。此外，该 EDO 还采用了一种特殊的多样性维持机制，即在每次在搜索结束后，评估各个子种群中最佳位置之间的距离，若该距离小于一个门限值，说明两个子种群可能覆盖在了同一片可行区域上，这样是没有意义的，所以算法会自动将适应度较差的那个子种群删除并随机初始化，以避免不必要的计算资源浪费。此基础上，本文又通过引入 CRA 与 RE 组件来将单纯的动态优化算法转变为针对性解决 ROOT 问题的方法。

3.3 鲁棒性估计机制 RE

RE 在每次环境变化后触发，其主要作用是估计各个有希望区域的鲁棒性。估计得到的鲁棒值被用于选择部署解，同时也在资源分配组件中起作用。该机制的伪代码描述见图 1：

Algorithm 2: Estimating the Robustness of pop_i (γ_i) and Managing the Explicit Archive $\mathcal{M}_i$

Input: $\mathcal{M}_i$ and $\vec{\mathbf{g}}_i^{*(t-1)}$.
Output: $\mathcal{M}_i$ and γ_i.

$\mathcal{M}_i \leftarrow \texttt{Push}(\vec{\mathbf{g}}_i^{*(t-1)})$;
$\mathrm{m}_i \leftarrow \texttt{Update}(\mathrm{m}_i)$;
$\gamma_i \leftarrow 0$;
$j \leftarrow t$;
repeat
  $j \leftarrow j - 1$;
  $\vec{\mathrm{y}} \leftarrow \texttt{Retrieve}(\vec{\mathbf{g}}_i^{*(j)} \in \mathcal{M}_i)$;
  if $f^{(t)}(\vec{\mathrm{y}}) \geq \mu$ **then**
    $\gamma_i \leftarrow \gamma_i + 1$;
until $\gamma_i = \mathrm{m}_i \vee f^{(t)}(\vec{\mathrm{y}}) < \mu$;
if $\gamma_i < \mathrm{m}_i$ **then**
  $\texttt{Remove}(\{\mathbf{g}_i^{*(t-k)} \in \mathcal{M}_i | k > \gamma_i\})$;
  $\mathrm{m}_i \leftarrow \gamma_i$;

图 1. RE 伪代码

其中每个子种群 pop_i 都拥有一个显式存档 M_i 为一个大小为 m_{max} 的循环队列，其中记录着每个子种群在历代环境中的最佳位置坐标 $\vec{g}_i^{*(t-1)}$，m_i 表示当前档案中的元素数量，当 $m_i = m_{max}$ 时档案已满，则会将档案中最老的解移除。每个子种群的鲁棒值 γ_i 的计算方法为，依次读取其档案 M_i 中的历史最佳位置坐标并代入当前环境的目标函数评估适应度，每有一个历史位置的当前环境适应度满足质量门槛，则 $\gamma_i + 1$，当出现第一个不满足的历史位置则停止循环，并将该位置及其之前环境的位置从档案中删除。通过这样的操作使得，拥有更多满足质量门槛的历史位置的子种群，其对应的鲁棒值 γ_i 也就越大，这也说明了其覆盖的区域可能拥有某些特质（例如，环境变换幅度不大）使得在其中找到鲁棒解的可能性更大。同时，由于对档案进行动态的删减，及时删去不满足质量门槛的解使得显式存档 M_i 不会随着时间推移变得十分臃肿庞大，这种修剪机制也有助于减少使用鲁棒性估计组件引起的适应度评估计算开销。

3.4 **计算资源分配机制** CRA

CRA 的目的是为了，在每次迭代前根据当前各个子种群的状态与其鲁棒性值，选择出最具潜力的子种群来运行优化算法，从而可以提升效率并降低不必要的计算开销。具体的做法是通过计算每个子种群覆盖的空间大小 λ，来反映子种群的状态。λ 为每个子种群中距离最远的两个个体的欧几里得距离。通过 λ 大小，将种群状态分为三档：

$$r_{min} < r_{cover} < r_{conv}$$

其中 $\lambda < r_{min}$ 表示子种群完全坍塌到一个极小区域;$\lambda < r_{cover}$ 表示子种群收缩到了一个可行域的峰值附近；$\lambda < r_{conv}$ 表示子种群收敛在一个可行域内。在第一个环境中，所有空间尺寸 $\lambda > r_{min}$ 的子种群都会被选中运行。在之后的环境中，则会如图 2伪代码所示过程进行：

Algorithm 3: Selecting Subpopulations to Run in the Current Iteration

Input: t, $f^{(t)}(\vec{\mathbf{s}})$, and γ and λ of all sub-populations.
Output: $\mathcal{L}$.

$\mathcal{L} \leftarrow \emptyset$; // Create an empty set $\mathcal{L}$
if $t = 1$ **then** // For the first environment
  foreach $\{\text{pop}_i \,|\lambda_i > r_{\min}\}$ **do**
    $\mathcal{L} \leftarrow \mathcal{L} \cup i$;
else
  $\gamma_{\max} = \max_{\{\forall \text{pop}_i\}}(\gamma_i)$;
  if $f^{(t)}(\vec{\mathbf{s}}) < \mu \wedge \{\exists \text{pop}_i \,| r_{\min} < \lambda_i \wedge \gamma_i = \gamma_{\max}\}$ **then**
    Mode $\leftarrow$ `Quick recovery`;
  else
    Mode $\leftarrow$ `Normal`;
  if Mode $=$ *Normal* **then**
    foreach $\{\text{pop}_i \,| r_{\min} < \lambda_i \leq r_{\text{cover}}\}$ **do**
      Calculate p_i using (10);
      if $\mathcal{U}[0, 1] \leq p_i$ **then**
        $\mathcal{L} \leftarrow \mathcal{L} \cup i$;
    foreach $\{\text{pop}_i \,| \lambda_i > r_{\text{cover}}\}$ **do**
      $\mathcal{L} \leftarrow \mathcal{L} \cup i$;
  else if Mode $=$ *Quick recovery* **then**
    foreach $\{\text{pop}_i \,| r_{\min} < \lambda_i \wedge \gamma_i = \gamma_{\max}\}$ **do**
      $\mathcal{L} \leftarrow \mathcal{L} \cup i$;

图 2. CRA 伪代码

首先 CRA 会根据当前系统的状态选择两种运行模式：1）快速恢复模式：当先前的部署解在本次环境中不再满足质量要求时，算法需要快速寻找一个新的解进行部署，此时运行快速恢复模式，选择所有 $r_{min} \leq \lambda$ 且拥有最大鲁棒值 γ_i 子种群运行，使算法专注于搜索最有希望的区域以确保找到好的替代解。2）正常模式：当先前部署的解不需要替换时，则优先选择 γ_i 值较大且 $r_{min} < \lambda_i < r_{cover}$ 子种群与所有 $\lambda_i > r_{cover}$ 的子种群，以节省计算资源。通过上面的状态划分，CRA 首先根据环境的情况选择从普通模式、快速恢复两种运行模式中选择一种。同时根据不同的模式将本次迭代中的计算资源分配给不同的子种群。通过使用这种策略，解决了 ROOT 问题中计算资源紧缺的问题，能够在部署解失效时及时寻找并部署新的鲁棒解，同时还保证了在环境变化中个子种群追踪到每个可行域。

3.5 决策机制

在经历环境变化后，先前部署的解可能在当前环境不再满足质量门槛，此时需要设计一个决策机制来选择部署一个新的解。本提出的决策机制为选择具有最大鲁棒值 γ_i 的子种群中，具有最佳适应度值的最佳位置 $\vec{g}_i^{*(t-1)}$，m_i，作为新的部署解。这么做的原因是，具有更大鲁棒值的区域可能拥有一些特殊的形态特征，使其更有希望选择出对即将到来的环境变化具有鲁棒性的解。

4 复现细节

4.1 与已有开源代码对比

本篇论文没有源代码，我选择在一个开源的进化动态优化测试平台 EDOLAB [3] 进行复现工作。该平台提供了，动态优化算法的基准测试问题，以及一系列实验测试框架与动态优化算法框架。在以上框架下，我通过阅读原论文中的伪代码与描述完成了对原论文所提出 ROOT 算法的复现。

4.2 实验环境搭建

本次代码复现的环境为 MATLAB R2023a-academic use 版本。如 3.2 节中所述，本次复现同原论文一样选用改进的多种群粒子群算法（PSO） [6] 做为整个算法的进化动态优化组件。同时，基准测试问题的选择与原论文一致选用 GMPB [7] 做为测试用例的生成器。

4.3 界面分析与使用说明

实验环境为 MATLAB R2023a。具体操作方法为，首先将 EDOLAB 的文件夹路径添加入 MATLAB 的当前操作路径，并且在图 3所示的代码段中，将 AlgorithmName 字段赋值为‘mPSO_RE_CRA’，并将 BenchmarkName 字段赋为‘GMPB’以运行所提出的算法并以 GMPB 为基准测试问题生成器。

```
36  %% ********Selecting Algorithm & Benchmark********
37  AlgorithmName = 'mPSO_RE_CRA';    %Please input the name of algorithm (EADO) you want to run here (names are case sensitive).
38  %  The list of algorithms (EADOs) and some of their details can be found in Table 1 of the EDOLAB's paper.
39  %  The current version of EDOLAB includes the following algorithms (EADOs):
40  %  'ACFPSO' , 'AMPDE' , 'AMPPSO' , 'AmQSO' , 'AMSO' , 'CDE' , 'CESO', 'CPSO' , 'CPSOR'
41  %  'DSPSO' , 'DynDE' , 'DynPopDE' , 'FTMPSO' , 'HmSO' ,  'IDSPSO' , 'ImQSO' , 'mCMAES'
42  %  'mDE' , 'mjDE' , 'mPSO' , 'mQSO' , 'psfNBC' , 'RPSO' , 'SPSO_AP_AD' , 'TMIPSO'
43  BenchmarkName = 'GMPB';     %Please input the name of benchmark you want to use here (names are case sensitive).
44  %  The current version of EDOLAB includes the following benchmark generators: 'MPB' , 'GMPB' , 'FPs'
```

图 3. 操作界面示意 1

之后可在图 4中设置有关 GMPB 生成问题的一些超参数。

```
编辑器 - D:\MATLAB\EDOLAB-MATLAB-main\EDOLAB-MATLAB-main\RunWithoutGUI.m *
+1  RunWithoutGUI.m *  |  main_mPSO_RE_CRA.m  |  SubPopulationGenerator_mPSO_RE_CRA.m  |  IterativeComponents_mPSO_RE_CRA.m
65  %% ********Benchmark parameters and Run number********
66  PeakNumber                  = 10;  %The default value is 10
67  ChangeFrequency             = 2500;%The default value is 5000
68  Dimension                   = 5;   %The default value is 5. It must be
69  ShiftSeverity               = 1;   %The default value is 1
70  EnvironmentNumber           = 100;  %The default value is 100
71  RunNumber                   = 31;   %It should be set to 31 in Experime
```

图 4. 操作界面示意 2

设置完毕后，点击 MATLAB 编辑器中的运行即可运行代码开始实验。

5 实验结果分析

实验的内容为，首先使用 GMPB 基准问题生成器生成数个环境 $f^{(t)}(\vec{x})$, 其实现了 $\mathbb{R}^d \mapsto \mathbb{R}^1$，即将输入的 d 维坐标转化为一个目标函数值（二维可视化的例子如图 5所示）。同时随着时间推移环境会发生变化，即切换 $f^{(t-1)}(\vec{x})$ 到 $f^{(t)}(\vec{x})$，实验的目标是找到在环境不断变化的情况下，其目标函数值依然能高于一个预定义的质量门槛的解（即坐标）。在实验过程中需要不断调用适应度评价函数对 PSO 算法中每个个体进行评价，即计算其对应坐标的目标函数值。从而实验中定义时间流逝的方式便是统计适应度评估的次数，当评估次数超过一个预定义的变化频率时，环境则发生变化。不同于一些动态优化问题算法需要自动检测环境变化，本次实验会显式地通知算法环境发生了变化，这么做的原因是因为现实中大部分动态问题会有各类传感器来检测环境变化，从而不需要算法层面的检测。

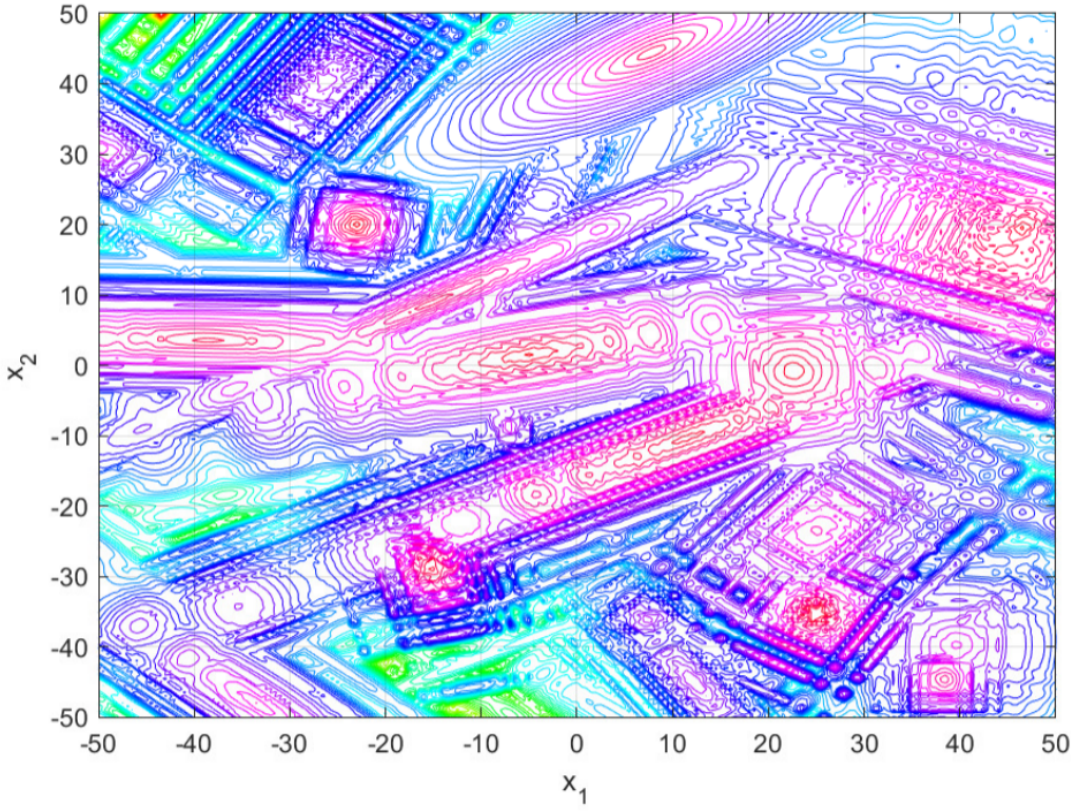

图 5. GMPB 产生的二维测试问题

实验的超参数设置与原文保持一致，具体需要设置的参数包括图 6所示测试问题相关的参数，用于设置 ROOT 问题的质量门槛、环境变化频率、计算资源预算等参数，以及 GMPB 生成测试问题间的变化程度与每个环境中移动峰的数量、问题维度等参数。下图左侧为原论文的参数设置，右侧为本次 MATLAB 实验中的参数设置。

Parameter	Symbol	Value(s)
Acceptability ROOT threshold	μ	40,45,50
Dimension	d	5,10
Numbers of promising regions	m	10,25,50,100
Shift severity	$\tilde{s}_i$ †	$\mathcal{U}[1,3]$,$\mathcal{U}[1,5]$,$\mathcal{U}[1,7]$ ‡
Change frequency	ϑ	250 ,500,1000,2500
Computational budget*	δ	$\lfloor \frac{\vartheta}{3} \rfloor$, $\lfloor \frac{\vartheta}{2} \rfloor$
Height severity	$\tilde{h}_i$	$\mathcal{U}[1,15]$
Width severity	$\tilde{w}_i$	$\mathcal{U}[0.1,1.5]$
Irregularity parameter τ severity	$\tilde{\tau}$	0.05
Irregularity parameter η severity	$\tilde{\eta}$	2
Angle severity	$\tilde{\theta}$	$\pi/9$
Search range	$[Lb, Ub]^d$	$[-50,50]^d$
Number of environments	T	100

```
Problem.QualityThreshold    = 40;%ROOT
Problem.ComputationalBudget = floor(ChangeFrequency/2);%ROOT
Problem.DeadLine            = 0;%ROOT
Problem.AverageSurvivalTime = 0;%ROOT
Problem.HeightSeverity      = 1+rand()*14;%ROOT
Problem.AngleSeverity       = pi/9;
Problem.TauSeverity         = 0.05;%ROOT
Problem.EtaSeverity         = 2;
Problem.WidthSeverity       = 0.1+rand()*1.4;%ROOT
%% ********Benchmark parameters and Run number********
PeakNumber                  = 10;
ChangeFrequency             = 2500;
Dimension                   = 5;
ShiftSeverity               = 1+rand()*4;
EnvironmentNumber           = 100;
RunNumber                   = 30;
```

图 6. 测试问题相关参数设置

算法相关的超参数设置如图 7所示，下图中上半部分为原论文的参数设置，下半部分为本次实验在 MATLAB 中的参数设置。具体内容包括 PSO 算法涉及到的参数与 EDO 组件需要的参数，如上文中提到的三个空间尺寸。

Method	Parameter	Value	Reference
PSO	χ	0.729843788	[10]
	C_1, C_2	2.05	[10]
	Neighbourhood topology	global star	[10]
	Sub-population size	5	[10]
mEDO framework*	φ_{excl}	$0.5\frac{Ub-Lb}{\sqrt[d]{p}}$ †	[9], [41]
	r_{conv}	$0.5\frac{Ub-Lb}{\sqrt[d]{p}}$	[9], [41] and Table S-II
Proposed components	$\mathfrak{m}_{\max}$	9	Table S-I
	r_{cover}	5	Fig. S-4
	$r_{\min}$	0.75	Fig. S-4

```
Optimizer.Dimension = Problem.Dimension;
Optimizer.PopulationSize = 5;
Optimizer.MaxCoordinate   = Problem.MaxCoordinate;
Optimizer.MinCoordinate = Problem.MinCoordinate;
Optimizer.FreeSwarmID = 1;%Index of the free swarm
Optimizer.ExclusionLimit = 0.5 * ((Optimizer.MaxCoordinate-Optimizer.MinCoordinate) / ((10) ^ (1 / Optimizer.Dimension)))
Optimizer.ConvergenceLimit = Optimizer.ExclusionLimit;
Optimizer.CoverLimit = 5;%ROOT
Optimizer.SleepLimit = 0.75;%ROOT
Optimizer.QuickRecoveryFlag = 0;%ROOT
Optimizer.DeployFlag = 1;%ROOT
Optimizer.DiversityPlus = 1;
Optimizer.x = 0.729843788;
Optimizer.c1 = 2.05;
Optimizer.c2 = 2.05;
Optimizer.SwarmNumber = 1;
Optimizer.MaxArchive = 9;
```

图 7. 测试问题相关参数设置

本次实验的评价指标为平均生存时间，计算方法为

$$\bar{S} = \frac{1}{t_{max}} \sum_{i=1}^{|S|} \sum_{j=0}^{n_i} j$$

其中 n_i 为每个解连续生存的环境数，$|S|$ 为所有部署解的集合，t_{max} 为总的环境数。该式衡量了一个算法找到的所有解能坚持的环境数量的平均值，该值越大则算法所找到的解的鲁棒性越强。

实验结果即为不同参数设置下独立 30 次运行所得到的 $\bar{S}$ 的均值与方差，原论文实验结果如图 8所示。本次复现在问题维度 $d = 5$ 10，质量门槛 μ 分别取 40、45、50，测试问题移动峰数 m 为 10、25 的情况下的实验，复现结果如图 9所示。对比两结果可以看出，复现结果与原论文结果相差不大。同时观察实验结果可以发现，固定其他条件不动时，随着质量门槛 μ 的提高，算法所找的解的平均生存时间逐渐下降，这是因为潜在的可行搜索空间被压缩。同时若固定其他条件，增大移动峰数量 m，解的平均生存时间增加，这是因为搜索空间中存在较多峰时，算法可以覆盖更多有希望的区域，缺失具有更高鲁棒性的解的区域的可能性会降低，进而提高了算法寻找更好的鲁棒解的性能。但不难发现原论文的实验在较低的维度下展开，同时算法找到的解的平均生存时间也并不长，哪怕在很简单的情况下也只是持续 23 代的时间，所以在 ROOT 领域中还存在很大的发展空间。

d	μ	m	Algorithms					
			$\text{mPSO}^{+\text{CRA}}_{+\text{RE}}$	mPSO	mPSO_{Rb}	mPSO_{Rsh}	PbM_{F}	PbM_{J}
5	40	10	2.14(0.25)	1.80(0.30)	1.90(0.33)	1.88(0.30)	0.14(0.03)	0.66(0.15)
		25	2.63(0.21)	1.83(0.18)	2.23(0.22)	2.21(0.24)	0.26(0.04)	1.10(0.24)
		50	2.99(0.16)	2.01(0.11)	2.35(0.19)	2.36(0.20)	0.33(0.05)	1.08(0.18)
		100	3.40(0.21)	2.42(0.26)	2.71(0.23)	2.65(0.26)	0.32(0.03)	1.03(0.17)
	45	10	1.41(0.12)	0.98(0.11)	1.01(0.11)	1.09(0.12)	0.07(0.02)	0.45(0.12)
		25	1.70(0.12)	1.23(0.15)	1.35(0.16)	1.40(0.16)	0.09(0.03)	0.63(0.13)
		50	1.90(0.10)	1.30(0.11)	1.48(0.14)	1.55(0.13)	0.07(0.01)	0.75(0.10)
		100	2.15(0.15)	1.52(0.16)	1.78(0.16)	1.74(0.15)	0.17(0.02)	0.71(0.11)
	50	10	0.85(0.09)	0.54(0.08)	0.62(0.09)	0.57(0.08)	0.01(0.00)	0.21(0.07)
		25	1.10(0.09)	0.70(0.10)	0.89(0.11)	0.85(0.10)	0.06(0.02)	0.29(0.07)
		50	1.27(0.07)	0.86(0.06)	1.02(0.09)	0.99(0.07)	0.06(0.02)	0.41(0.05)
		100	1.46(0.10)	0.98(0.11)	1.18(0.13)	1.15(0.10)	0.09(0.03)	0.35(0.09)
10	40	10	1.06(0.08)	0.82(0.09)	0.83(0.09)	0.83(0.09)	0.07(0.02)	0.38(0.07)
		25	1.35(0.12)	1.03(0.11)	1.04(0.11)	1.01(0.10)	0.07(0.03)	0.45(0.11)
		50	1.27(0.13)	0.92(0.15)	0.94(0.15)	0.96(0.15)	0.07(0.02)	0.41(0.14)
		100	1.60(0.18)	0.90(0.14)	0.98(0.14)	0.97(0.14)	0.11(0.03)	0.56(0.16)
	45	10	0.65(0.06)	0.42(0.05)	0.47(0.06)	0.46(0.05)	0.02(0.01)	0.21(0.05)
		25	0.80(0.08)	0.51(0.07)	0.56(0.07)	0.56(0.07)	0.02(0.01)	0.25(0.07)
		50	0.82(0.09)	0.49(0.10)	0.52(0.10)	0.53(0.10)	0.06(0.01)	0.25(0.10)
		100	0.91(0.08)	0.47(0.05)	0.47(0.05)	0.47(0.05)	0.06(0.01)	0.33(0.09)
	50	10	0.30(0.02)	0.18(0.02)	0.19(0.02)	0.18(0.02)	0.02(0.01)	0.10(0.02)
		25	0.37(0.04)	0.23(0.03)	0.25(0.04)	0.26(0.03)	0.00(0.00)	0.12(0.03)
		50	0.40(0.06)	0.19(0.05)	0.21(0.05)	0.20(0.05)	0.00(0.00)	0.10(0.06)
		100	0.38(0.05)	0.18(0.03)	0.18(0.03)	0.18(0.03)	0.00(0.00)	0.13(0.04)

图 8. 原论文实验结果

d	μ	m	平均生存时间（方差）
5	40	10	2.32（0.22）
		25	2.72（0.25）
	45	10	1.23（0.19）
		25	1.50（0.14）
	50	10	0.65（0.12）
		25	0.92（0.11）
10	40	10	1.12（0.12）
		25	1.21（0.15）
	45	10	0.43（0.10）
		25	0.63（0.14）
	50	10	0.24（0.08）
		25	0.38（0.03）

图 9. 复现实验结果

6 总结与展望

本篇实验报告根据 Yazdani 等人 [8] 发表在 TEVC 2023 上关于 ROOT 的工作，做了系统的介绍与代码复现工作，并且对实验平台与实验结果进行了介绍与分析。同时根据实验结果显示，现阶段的 ROOT 方法还是比较初级的，其在 ROOT 问题上并未取得非常好的效果，以至于现阶段的算法性能与现实需求存在脱节。所以该领域还存在比较大的研究空间，还可以尝试结合更多先进的深度学习模型来对变化的环境与环境中的不确定性进行建模，从而可以更全面地描绘系统变化，有助于找到鲁棒解。同时，ROOT 领域广泛使用的评价指标平均生存时间并不能十分良好地反映算法的鲁棒性，可能存在平均生存时间长的算法，其切换部署解的次数比平均生存时间短的解要更多，这背离了原 ROOT 问题想要降低切换部署解次数的初衷，所以一个能够客观反映算法在 ROOT 问题上的表现的性能指标也是有待研究的课题。总的来说，ROOT 问题的研究集中于动态环境中解决方案的稳健性，特别关注于那些不适合频繁更换策略的场景。近年来的研究取得了开发出许多方法来评估和提高解决方案在不断变化的条件下的鲁棒性。未来的研究可能包括更精确的环境预测模型、算法的适应性提升，以及对不同类型动态问题的应用扩展。这些努力将进一步提高算法在面对未知和不确定环境变化时的效率和有效性。

参考文献

[1] Haobo Fu, Bernhard Sendhoff, Ke Tang, and Xin Yao. Finding Robust Solutions to Dynamic Optimization Problems, page 616–625. Jan 2013.

[2] Yaochu Jin, Ke Tang, Xin Yu, Bernhard Sendhoff, and Xin Yao. A framework for finding robust optimal solutions over time. Memetic Computing, 5(1):3–18, Mar 2013.

[3] Mai Peng, Zeneng She, Delaram Yazdani, Danial Yazdani, Wenjian Luo, Changhe Li, Juergen Branke, Trung Nguyen, Amir Gandomi, Yaochu Jin, and Xin Yao. Evolutionary dynamic optimization laboratory: A matlab optimization platform for education and experimentation in dynamic environments, 08 2023.

[4] Danial Yazdani, Jürgen Branke, MohammadNabi Omidvar, TrungThanh Nguyen, and Yao Yao. Changing or keeping solutions in dynamic optimization problems with switching costs. Genetic and Evolutionary Computation Conference,Genetic and Evolutionary Computation Conference, Jul 2018.

[5] Danial Yazdani, Trung Thanh Nguyen, and Jurgen Branke. Robust optimization over time by learning problem space characteristics. IEEE Transactions on Evolutionary Computation, 23(1):143–155, Feb 2019.

[6] Danial Yazdani, Mohammad Nabi Omidvar, Jurgen Branke, Trung Thanh Nguyen, and Xin Yao. Scaling up dynamic optimization problems: A divide-and-conquer approach. IEEE Transactions on Evolutionary Computation, 24(1):1–15, Feb 2020.

[7] Danial Yazdani, MohammadNabi Omidvar, Donya Yazdani, Jürgen Branke, Thanh Nguyen, AmirH Gandomi, Yaochu Jin, Xin Yao, Danial Yazdani, Donya Yazdani, Juergen Branke, and TrungThan Nguyen. Robust optimization over time: A critical review.

[8] Danial Yazdani, Donya Yazdani, Jürgen Branke, Mohammad Nabi Omidvar, Amir Hossein Gandomi, and Xin Yao. Robust optimization over time by estimating robustness of promising regions. IEEE Transactions on Evolutionary Computation, page 657–670, Jun 2023.

[9] Xin Yu, Yaochu Jin, Ke Tang, and Xin Yao. Robust optimization over time —a new perspective on dynamic optimization problems. pages 1 – 6, 08 2010.