

# 题目

## A Three-Level Radial Basis Function Method for Expensive Optimization

### 基于三层径向基函数的昂贵优化算法

#### 摘要

本文提出了一种三层径向基函数（TLRBF）辅助优化算法，用于昂贵的优化。它由三个搜索过程组成：1）全局探索搜索是在整个搜索空间中，通过优化受距离约束的全局RBF近似函数来求解；2）子区域搜索是通过在模糊聚类确定的子区域中最小化RBF近似函数来生成解；3）局部开发搜索是通过求解当前最佳解附近的局部RBF近似模型来生成解。与其他一些最先进的算法相比，在5个常用的可扩展基准问题、10个计算成本CEC2015问题以及一个实际翼型设计优化问题上，我们提出的算法在昂贵的优化方面表现良好。

关键词： RBF模型；模糊聚类

## 1 引言

在许多工程优化问题，评估候选解的质量需要非常昂贵的计算机或物理仿真[ 1 ]。这些问题被称为黑箱昂贵优化问题。由于无法获得问题的解析表达式，传统的数学优化方法无法直接用于求解这类问题。经典的进化算法( Evolutionary Algorithms, EAs )也不适用于昂贵的优化问题，因为它们通常需要大量的函数评估。为了处理昂贵的优化问题，代理辅助优化算法( SAOs )，包括传统的SAOs [ 2 ]和代理辅助进化算法( SAEAs ) [ 3 ]，已经被开发出来。SAOs基于代理模型建模和预测比精确评估在计算上更便宜的假设，采用代理模型进行选择在搜索过程中进行精确评估的解决方案。在SAOs中使用的代表性代理模型包括多项式回归模型、径向基函数模型( RBFs )、高斯过程( GPs )、支持向量机和人工神经网络。

已经开发了一些基于代理模型选择解决方案的填充采样准则。例如，在使用GPs时，高效的全局优化算法[ 4 ]选择期望改进最大的候选解进行评估。其他准则，如改善概率[ 5 ]、置信下限[ 6 ]、最大方差[ 7 ]等也得到了广泛的应用。这些准则在每次迭代中只选择一个候选解。缺点是很难很好地平衡勘探和开采[ 8 ]。除了传统的SAOs，SAEAs也被开发用于解决各种昂贵的优化问题。一些SAEAs [ 9 ] ~ [ 13 ]仅使用代理模型对遗传算子产生的候选解进行预筛选。他们并不直接使用代理来产生新的候选解。因此，当计算资源非常有限时，这些算法是低效的[ 14 ] - [ 16 ]。Wang等[ 17 ]利用基于委员会的主动学习提出了一种新的模型管理策略，其中全局模型管理策略使用集成代理模型来定位最不确定和最佳评估解决方案，而局部模型管理策略用于搜索局部代理模型的最佳评估解决方案。与预筛选机制相比，模型管理策略可以更有效地利用代理模型。

一般而言，构建代理模型的计算开销会随着训练数据规模的增大而增大[ 18 ]。因此，如何充分利用可获得的数据来指导优化过程是非常具有挑战性的。针对这一问题，本文提出了一种三级RBF ( TLRBF )辅助优化算法。具体来说，为了在整个搜索空间中集成粗粒度的全

局搜索 ( global search, GS ), 在有希望的子区域中集成中粒度的子区域搜索 ( subarea search, SS ), 在最有希望的局部区域中集成细粒度的局部搜索 ( local search, LS ), 在TLRBF的每次迭代中, 分别生成GS、SS和LS评估的3个解。在GS中, 所有可用解被用于构建全局RBF近似函数, 然后使用随机抽样方法提供一个新的解决方案, 该解决方案具有良好的目标函数值, 并与先前评估的解决方案保持一定的距离。在SS中, 由模糊聚类确定的可用解的子集是用于训练一个子区域RBF近似函数, 然后通过优化器对其进行优化, 以找到其最优值, 从而进行精确评估。在LS中, 当前最优解及其邻域被用来构建局部RBF近似函数, 并通过优化器最小化该函数, 以寻找其最优值进行评估。

## 2 相关工作

### 2.1 黑箱昂贵优化问题

本文考虑如下黑箱昂贵优化问题:

$$\begin{aligned} & \text{minimize } f(\mathbf{x}) \\ & \text{subject to } \mathbf{x} \in \Omega \end{aligned}$$

其中 $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ 是一个包含 $d$ 个决策变量的向量,  $d$ 是搜索空间 $\Omega = [a, b]$ 的维数, 其中向量 $a$ 和 $b$ 分别是搜索空间的下界和上界。 $f(\mathbf{x})$ 是一个标量目标函数。我们假设 $f(\mathbf{x})$ 的精确估计是昂贵的, 并且无法获得 $f(\mathbf{x})$ 的数学表达式。

### 2.2 RBF模型

RBF模型[19]已成为插值散乱数据最流行的技术之一。给定一组不同的中心 $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k\}$ , RBF插值模型可以表示为

$$S(\mathbf{x}) = \sum_{k=1}^K w_k \varphi(\|\mathbf{x} - \mathbf{x}_k\|)$$

式中:  $\mathbf{W} = (w_1, w_2, \dots, w_K)^T$ 为权重向量,  $\varphi: \mathbb{R} \rightarrow \mathbb{R}$ 为实值基函数。

对于给定的训练数据集 $D = (\{\mathbf{x}_i, f(\mathbf{x}_i)\}, i = 1, 2, \dots, N)$ , 其中 $N$ 为训练数据集的大小,  $\mathbf{x}_i \in \mathbb{R}^d$ 和 $f(\mathbf{x}_i) \in \mathbb{R}$ 分别表示输入和对应的输出。假设训练数据集 $D$ 中的所有点都是中心, 为了得到权重向量 $\mathbf{W}$ , 需要满足以下插值条件:

它可以写成一个矩阵的形式:  $\Phi \mathbf{W} = \mathbf{f}$

$$\Phi_{ij} = \varphi(\|\mathbf{x}_i - \mathbf{x}_j\|), i, j = 1, 2, \dots, N, \mathbf{f} = (f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_N))^T.$$

最后, 通过求解线性方程组(4)可以得到权重向量 $\mathbf{W}$ , 如下:

$$\mathbf{W} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{f}.$$

有几类基函数可以保证矩阵的非奇异性, 如高斯函数、样条函数、多二次函数等。在本文中, 下面的多二次函数被用作基函数:

$$\varphi(r) = \sqrt{r^2 + \delta^2}$$

式中:  $\delta$  为基函数的形状参数。最后, 对于一个新的输入 $\mathbf{x}^*$ , 其预测输出可以计算为

$$S(\mathbf{x}^*) = \sum_{k=1}^N w_k \varphi(\|\mathbf{x}^* - \mathbf{x}_k\|)$$

在本文中，RBF模型和GP模型是使用MATLAB的替代模型工具箱 $toolbox^2$  [ 20 ]实现的。基于作者的初步实验(见补充文件表S . I)，RBF比GP更适合作者提出的三层框架。因此，本文采用RBF模型。此外，RBF模型在本文中使用的所有测试问题上的逼近误差在图中进行了说明。

### 3 本文方法

#### 3.1 本文方法概述

此部分对本文将要改进的工作进行概述，算法整体框架如图 1所示：

---

**Algorithm 1** TLRBF

---

```

1:  Generate an initial training data set  $\mathbf{D}_1$ 
2:  Set the current database  $\mathbf{D} = \mathbf{D}_1$ 
3:  while the stopping criterion is not satisfied
4:     $\mathbf{D} = \text{GS}(\mathbf{D}, \Omega, \alpha, m)$  // Global level
5:     $\mathbf{D} = \text{SS}(\mathbf{D}, N, L_1, L_2, \Delta_S)$  // Medium level
6:     $\mathbf{D} = \text{LS}(\mathbf{D}, k, \Delta_L)$  // Local level
7:  end
Output: The best solution  $\mathbf{x}_b$  in database  $\mathbf{D}$ 

```

---

图 1. 方法示意图

算法1给出了TLRBF的框架。首先，通过一些实验设计方法在整个搜索空间中生成一组初始样本 $\mathbf{x}_{\text{init}} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n_0}\}$ ，其中 $n_0$ 为初始样本的个数。本文采用拉丁超立方抽样(LHS)方法产生初始样本，通过对初始样本进行评估，得到初始训练数据集 $\mathbf{D}_1 = \{(\mathbf{x}_i, f_i), i = 1, 2, \dots, n_0\}$ 。初始化后，算法迭代更新 $\mathbf{D}$ ，直到满足停止准则。在TLRBF的每次迭代中，在全局层面，利用数据库 $\mathbf{D}$ 构建全局RBF模型 $S_G(\mathbf{x})$ ， $\text{GS}$ 在 $S_G(\mathbf{x})$ 的基础上，在整个搜索空间中搜索一个有希望的未探索解 $\mathbf{x}^G$ ，用于代价昂贵的评估。在中观层面，基于模糊聚类选择当前数据库 $\mathbf{D}$ 的子集 $\mathbf{DS}$ ，建立子区域RBF模型 $S_S(\mathbf{x})$ 。在 $S_S(\mathbf{x})$ 的基础上， $\text{SS}$ 通过最小化 $S_S(\mathbf{x})$ 在子搜索空间 $\mathbf{S}$ 中找到一个有希望的解 $\mathbf{x}^S$ 。在局部层面，利用当前最优解 $\mathbf{x}_b$ 及其 $k$ 个近邻点建立局部RBF模型 $S_L(\mathbf{x})$ ，并在相应的 $\text{LS}$ 空间 $\mathbf{L}$ 中优化 $S_L(\mathbf{x})$ 得到新解 $\mathbf{x}^L$ 。当满足停止准则时，数据库 $\mathbf{D}$ 中的最优解 $\mathbf{x}_b$ 将作为最终解。

---

**Algorithm 2**  $\text{GS}(\mathbf{D}, \Omega, \alpha, m)$

---

**Input:** Database  $\mathbf{D}$ , search space  $\Omega$ , scale factor  $\alpha$ , random sample size  $m$

```

1:  if the stopping criterion is not satisfied
2:    Use  $\mathbf{D}$  to construct the global RBF model  $S_G(\mathbf{x})$ 
3:    Generate set  $\mathbf{A} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_m\}$  in  $\Omega$ 
4:    Calculate  $\Delta_G = \max_{\mathbf{y} \in \mathbf{A}} \min_{\mathbf{x} \in \mathbf{D}} \|\mathbf{y} - \mathbf{x}\|$ 
5:    Set  $\mathbf{B} = \{\mathbf{y} | \mathbf{y} \in \mathbf{A}, \min_{\mathbf{x} \in \mathbf{D}} \|\mathbf{y} - \mathbf{x}\| > \alpha \Delta_G\}$ 
6:     $\mathbf{x}^G = \arg\min_{\mathbf{y} \in \mathbf{B}} S_G(\mathbf{y})$ 
7:  end
8:  Conduct evaluation to obtain  $f(\mathbf{x}^G)$ 
9:  Update database by  $\mathbf{D} = \mathbf{D} \cup (\mathbf{x}^G, f(\mathbf{x}^G))$ 
Output: Database  $\mathbf{D}$ 

```

---

在 $\text{GS}$ 中，需要生成一个大的样本集来覆盖整个搜索空间。随机均匀抽样法和一些试验设计方法如LHS等均可用于此目的。然而，我们的实验结果(见补充文件表S .III)表明，采样策

略对所提算法的性能并不关键。而且，与随机均匀采样方法相比，使用DOE方法生成大样本集本身就是一个耗时的过程(见补充文件表S . IV)。因此，在实验中，作者采用随机均匀采样的方法。

---

**Algorithm 3** SS( $\mathbf{D}, N, L_1, L_2, \Delta_S$ )

---

**Input:** Database  $\mathbf{D}$ ,  $N$ ,  $L_1$ ,  $L_2$ ,  $\Delta_S$

```

1: if the stopping criterion is not satisfied
2:   if  $N \leq L_1$ 
3:      $\mathbf{D}_S = \mathbf{D}$ 
4:   else
5:     Normalize the database  $\mathbf{D}$ 
6:     Calculate  $c$  as (9)
7:     Cluster the database  $\mathbf{D}$  into  $c$  groups based on
       fuzzy  $c$ -means clustering
8:     Calculate the quality of each cluster as (10)
9:     Rank all the clusters as (11)
10:    Calculate the selection probability as (12)
11:    Randomly generate an integer  $r \in [1, c]$ 
12:    while  $rand > p_r$ 
13:      Randomly generate an integer  $r \in [1, c]$ 
14:    end
15:     $\mathbf{D}_S = \mathbf{C}_r$ 
16:  end
17:  Use  $\mathbf{D}_S$  to construct a subregion RBF  $S_S(\mathbf{x})$ 
18:  Obtain  $\mathbf{x}^S$  by using JADE to solve (13)
19:  if  $\min_{\mathbf{x} \in \mathbf{D}} \|\mathbf{x}^S - \mathbf{x}\| \geq \Delta_S$ 
20:    Compute  $f(\mathbf{x}^S)$ 
21:    Update database  $\mathbf{D}$  by  $\mathbf{D} = \mathbf{D} \cup (\mathbf{x}^S, f(\mathbf{x}^S))$ 
22:  end
23: end

```

**Output:** Database  $\mathbf{D}$

---

算法3总结了SS的伪代码。需要指出的是，如果解 $\mathbf{x}^S$ 过于接近一个先前评估的点，即 $\min_{\mathbf{x} \in \mathbf{D}} \|\mathbf{x}^S - \mathbf{x}\| < \Delta_S$ ，其中 $\Delta_S$ 是一个很小的正的常数，则 $\mathbf{x}^S$ 将被直接移除。综上，SS的优点如下：1)模糊聚类能够将整个搜索空间划分为多个重叠的平衡子区域，从而提高边界区域的逼近精度[41]；2)较好的子区域具有较高的开采概率，这自然能够在勘探和开采之间取得良好的平衡；3)下一次代理模型在 $\mathbf{x}^S$ 附近的近似质量将得到改善，为后续迭代中利用 $\mathbf{x}^S$ 附近的区域带来了机会。

---

**Algorithm 4** LS( $\mathbf{D}, k, \Delta_L$ )

---

**Input:** Database  $\mathbf{D}$ , neighbor size  $k$ ,  $\Delta_L$

```

1: if the stopping criterion is not satisfied
2:   Identify the current best solution  $\mathbf{x}_b$ 
3:   Determine the local data set  $\mathbf{D}_L$ 
4:   Use  $\mathbf{D}_L$  to construct the local RBF  $S_L(\mathbf{x})$ 
5:   Obtain  $\mathbf{x}^L$  by using JADE to solve (15)
6:   if  $\min_{\mathbf{x} \in \mathbf{D}} \|\mathbf{x}^L - \mathbf{x}\| \geq \Delta_L$ 
7:     Compute  $f(\mathbf{x}^L)$ 
8:     Update database  $\mathbf{D}$  by  $\mathbf{D} = \mathbf{D} \cup (\mathbf{x}^L, f(\mathbf{x}^L))$ 
9:   end
10: end

```

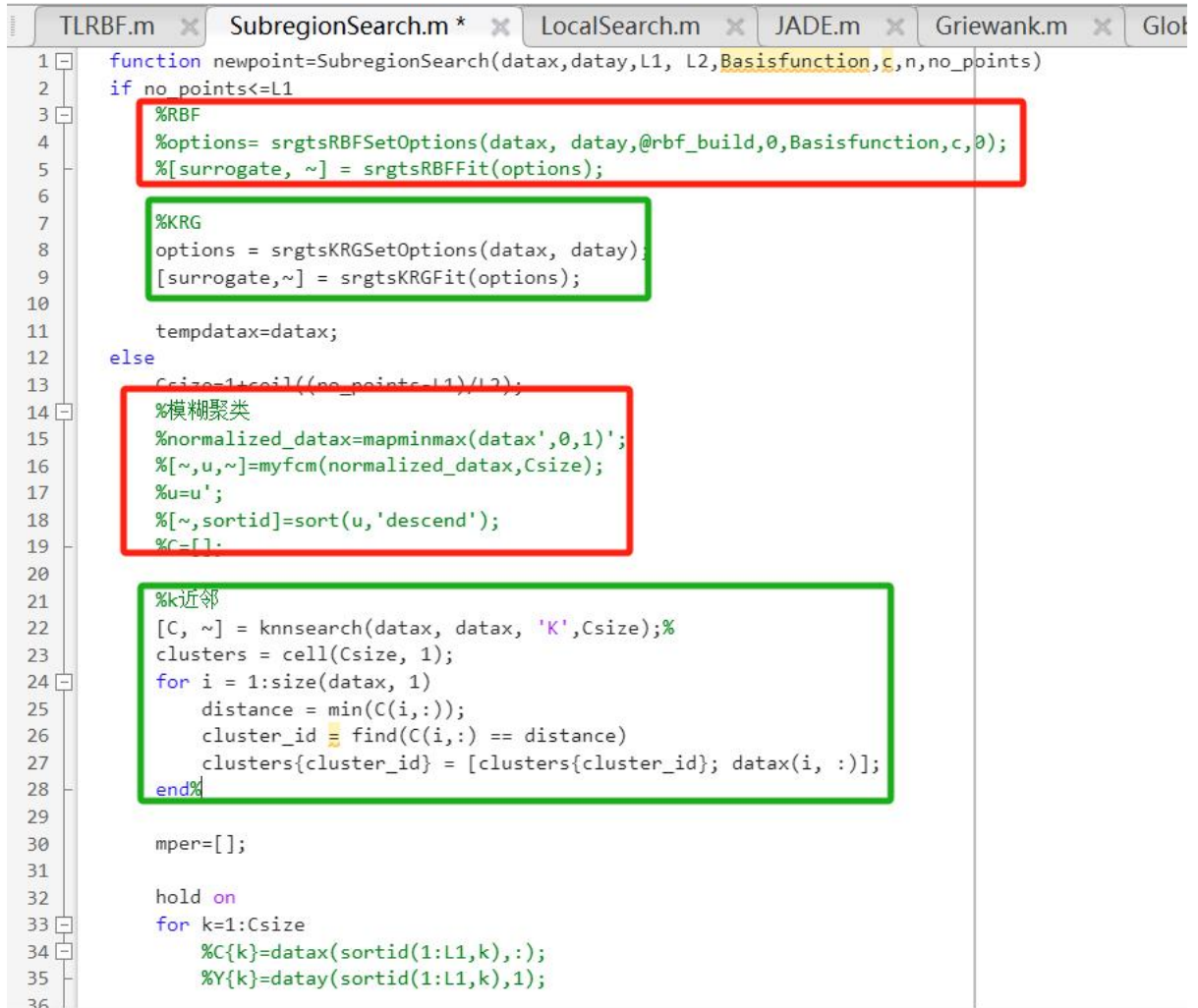
**Output:** Database  $\mathbf{D}$

---

算法4给出了LS的伪代码。类似地，只有满足距离约束 $\min_{\mathbf{x} \in D} \|\mathbf{x}^S - \mathbf{x}\| \geq \Delta_L$ ，才能计算得到 $\mathbf{x}^L$ ，其中 $\Delta_L$ 也是一个很小的正的常数。在SS和LS中， $\Delta_S$ 和 $\Delta_L$ 被用来保持解的多样性。 $\Delta_L$ 应小于 $\Delta_S$ 。

## 4 复现细节

### 4.1 与已有开源代码对比



```
1 function newpoint=SubregionSearch(datax,datay,L1, L2,Basisfunction,c,n,no_points)
2 if no_points<=L1
3     %RBF
4     %options= srgtsRBFSetOptions(datax, datay,@rbf_build,0,Basisfunction,c,0);
5     %[surrogate, ~] = srgtsRBFFit(options);
6
7     %KRG
8     options = srgtsKRGSetOptions(datax, datay);
9     [surrogate,~] = srgtsKRGFit(options);
10
11     tempdatax=datax;
12 else
13     Csize=1+ceil((no_points-L1)/L2);
14     %模糊聚类
15     %normalized_datax=mapminmax(datax',0,1)';
16     %[,u,~]=myfcm(normalized_datax,Csize);
17     %u=u';
18     %[,sortid]=sort(u,'descend');
19     %C=[];
20
21     %k近邻
22     [C, ~] = knnsearch(datax, datax, 'K',Csize);%
23     clusters = cell(Csize, 1);
24     for i = 1:size(datax, 1)
25         distance = min(C(i,:));
26         cluster_id = find(C(i,:) == distance)
27         clusters{cluster_id} = [clusters{cluster_id}; datax(i, :)];
28     end%
29
30     mper=[];
31
32     hold on
33     for k=1:Csize
34         %C{k}=datax(sortid(1:L1,k),:);
35         %Y{k}=datay(sortid(1:L1,k),1);
36
```

红色方框里的两部分分别是本文的开源代码，构建局部的RBF模型和使用的模糊聚类方法；绿色方框里的是替换了局部的RBF模型为KRG模型，以及使用了k近邻聚类方法。



```

30 mper=[];
31
32 hold on
33 for k=1:Csize
34     %C{k}=datax(sortid(1:L1,k),:);
35     %Y{k}=datay(sortid(1:L1,k),1);
36
37     %k近邻
38     matrix_array=cellfun(@(x) x,clusters, 'UniformOutput', false);
39     B=matrix_array{k};
40     y_predrbf = ackley(B);
41     %y_predrbf = ellipsoid(B);
42     %y_predrbf = rosenbrock(B);
43     %y_predrbf=griewank(B);
44     %y_predrbf=rastrigin(B);
45
46     %mper(k)=mean(Y{k});
47     mper(k)=mean(y_predrbf);
48
49     %RBF
50     %options= srgtsRBFSetOptions(C{k}, Y{k},@rbf_build,0,Basisfunction,c,0);
51     %模糊聚类
52     %options= srgtsRBFSetOptions(B,y_predrbf,@rbf_build,0,Basisfunction,c,0);
53     %k近邻
54
55     %[Surrogate{k}, State{k}] = srgtsRBFfit(options);
56     %surrogate=Surrogate{k};
57
58     %KRG
59     options = srgtsKRGSetOptions(B,y_predrbf);
60     [Surrogate{k},~] = srgtsKRGfit(options);
61     surrogate=Surrogate{k};
62
63
64 end

```

红色方框中是本文的开源代码，蓝色方框是跟k近邻相关的，绿色的是跟局部模型KRG相关的代码。

```

% SubregionSearch.m
function newpoint=localsearch(datax,datay,basisfunction,c,n,k)
%...
%RBF
%options= srgtsRBFSetOptions(local_datax, local_datay,@rbf_build,0,Basisfunction,c,0);
%[surrogate, ~] = srgtsRBFfit(options);
%KRG
options= srgtsKRGSetOptions(local_datax, local_datay);
[surrogate, ~] = srgtsKRGfit(options);
newpoint=JADE(lu,surrogate,100,100,n);

% LocalSearch.m
function v=JADE(lu,surrogate,popszie,gmax,n)
%...
%valParents=srgtsRBFEvaluate(popold, surrogate);
valParents=srgtsKRGEvaluate(popold, surrogate);

```

接着要将这几个文件中的RBF模型替换为KRG。

## 4.2 实验环境搭建

MATLAB R2022b.

## 4.3 创新点

KNN算法可以适应非线性分类，而模糊聚类则主要适用于线性分类。对于优化问题中的复杂种群划分，KNN的适应性更强。所以将模糊聚类替换成了k近邻聚类。克里金的预测能力较强，所以把局部的RBF代理模型替换成KRG代理模型进行了对比。

## 5 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。

Problem	D	TLRBF	TLRBF (K近邻)	TLRBF (K近邻+KRG)
Ellipsoid	5	<b>7.2023E-05</b>	1.2069E-02	2.2490E-01
	10	1.3583E+01	<b>5.1540E+00</b>	2.4409E+01
	20	<b>4.2528E+00</b>	5.5527E+02	1.7917E+02
Rosenbrock	5	9.5357E-01	0	0
	10	7.1611E+00	0	0
	20	7.8993E+01	0	0
Ackley	5	5.5530E+00	3.5958E+00	<b>2.9261E+00</b>
	10	8.7689E+00	3.5751E+00	<b>2.8578E+00</b>
	20	7.5504E+00	<b>3.5747E+00</b>	3.5944E+00
Griewank	5	<b>1.9099E-01</b>	2.0923E-01	5.0139E-01
	10	1.0583E+00	6.5340E-01	<b>4.6017E-01</b>
	20	1.7898E+00	<b>4.2094E-01</b>	7.9619E-01
Rastrigin	5	<b>1.9902E+00</b>	4.9812E+00	4.9941E+00
	10	1.7915E+01	<b>9.9691E+00</b>	9.9693E+00
	20	5.0755E+01	<b>1.9902E+01</b>	1.9982E+01
Rank		2.25	1.75	2

图 3. 实验结果示意

五个测试函数，三个维度（5d、10d、20d），TLRBF是本文的开源代码，TLRBF（k近邻）是将本文的局部搜索中使用到的模糊聚类替换为了k近邻算法，TLRBF（k近邻+KRG）是将模糊聚类替换为了k近邻，还将局部搜索的RBF模型替换为了KRG模型。实验结果中，每一行中三个数据进行对比，蓝色加粗的是最好的，都是相同的初始种群分别运行了10次，可以看到后面两种都较本文有些许改进。

## 6 总结与展望

目前实验中的一些不足是跑的次数不够多，如果能都跑30次结果就更有说服力，不过跑着太费时间了。所以一些的优化方向可能是在不降低精度的基础上降低程序的运行时间，比如如何去减少构建一些代理模型，构建的代理模型如何更快的去找到最优解等等。