基于多种优化算法的有限地图最低海拔搜索策略比较研究

作者信息

姓名： 鲁益杰

学校： 四川大学

学院： 数学学院

专业： 数学与智能科学双学士学位

学号： 2025141210006

日期： 2025 年 12 月 18 日

摘要

本文研究在有限但未完全勘探的地图中寻找海拔最低点的问题，目标是在不遍历所有点的前提下，通过多种优化算法尽可能找到全局最低点。传统遍历方法成本高昂，因此本文采用三种不同的优化策略进行高效搜索：模拟退火算法 (SA)、基于高斯过程的贝叶斯优化算法 (GP-BO) 和基于插值模型的序列决策算法 (SDIM)。**本文的创新点在于**：1) 首次系统比较三种算法在多种地形中的表现；2) 开发了完整的可视化系统，能够实时展示地形生成、算法搜索过程和结果分析；3) 提供了可直接运行的 Python 代码，实现从地形生成到结果可视化的全流程自动化。实验结果表明，不同算法在不同地形中表现各异，为实际应用中的算法选择提供了科学依据。

关键词：模拟退火；贝叶斯优化；高斯过程；序列决策；地形搜索；可视化分析

目录

1 引言

1.1 研究背景与意义

在现代勘探、机器人导航和环境监测等领域，经常需要在未知地形中寻找极值点。这类问题的核心挑战在于信息的有限性：地形信息仅在探索点处可知，而全面探索的成本过高。因此，开发能够在有限探索次数下高效找到全局最优的算法具有重要的理论价值和实践意义。

可视化在算法研究中的重要性：地形搜索问题本质上是一个空间优化问题，直观的可视化能够帮助研究者理解算法行为、分析问题特征、验证算法有效性。本文特别强调可视化分析，通过多维度的可视化技术，全面展示算法的搜索过程、收敛特性和性能表现。

1.2 研究内容与创新点

本文的主要研究内容和创新点包括：

1. **多算法系统比较：**首次同时对比模拟退火、贝叶斯优化和序列决策三种算法在地形搜索问题中的性能。
2. **完整可视化系统：**开发了包含地形生成、算法搜索、结果分析的全流程可视化工具。
3. **交互式实验平台：**提供可直接运行的 Python 代码，支持参数调整和实时可视化。
4. **多维度性能评估：**从精度、效率、收敛性、鲁棒性等多个维度评估算法性能。

2 相关算法介绍

2.1 模拟退火算法（Simulated Annealing, SA）

模拟退火算法是一种受固体退火过程启发的全局优化算法。它通过引入”温度”参数来控制搜索过程，在高温阶段接受较差的解以扩大搜索范围，随着温度降低逐渐收敛到最优解附近。该算法特别适用于解决复杂的组合优化问题，能够有效避免陷入局部最优。

Algorithm 1 模拟退火算法用于地形最低点搜索

Require: 地图范围 $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$，初始温度 T_{initial}，终止温度 T_{final}，冷却系数 α $(0 < \alpha < 1)$，每个温度下的迭代次数 L

Ensure: 找到的最低点位置 $(x_{\text{best}}, y_{\text{best}})$ 及其海拔 h_{best}

初始化：
  在当前地图范围内随机选择初始点 $(x_{\text{current}}, y_{\text{current}})$
  获取该点海拔 $h_{\text{current}} \leftarrow \text{GetElevation}(x_{\text{current}}, y_{\text{current}})$
  $(x_{\text{best}}, y_{\text{best}}) \leftarrow (x_{\text{current}}, y_{\text{current}})$
  $h_{\text{best}} \leftarrow h_{\text{current}}$
  $T \leftarrow T_{\text{initial}}$
while $T > T_{\text{final}}$ **do**
  for $i = 1$ to L **do**
    生成邻域解：
    $(x_{\text{new}}, y_{\text{new}}) \leftarrow \text{GenerateNeighbor}(x_{\text{current}}, y_{\text{current}}, T)$
    边界处理：
    $(x_{\text{new}}, y_{\text{new}}) \leftarrow \text{ConstrainToMap}(x_{\text{new}}, y_{\text{new}})$
    获取新点海拔：
    $h_{\text{new}} \leftarrow \text{GetElevation}(x_{\text{new}}, y_{\text{new}})$
    计算海拔变化（寻找最低点，所以变化为负表示改进）：
    $\Delta h \leftarrow h_{\text{new}} - h_{\text{current}}$
    决定是否接受新解：
    if $\Delta h < 0$ **then**
      新点海拔更低，总是接受
      $(x_{\text{current}}, y_{\text{current}}) \leftarrow (x_{\text{new}}, y_{\text{new}})$
      $h_{\text{current}} \leftarrow h_{\text{new}}$
      更新历史最优：
      if $h_{\text{new}} < h_{\text{best}}$ **then**
        $(x_{\text{best}}, y_{\text{best}}) \leftarrow (x_{\text{new}}, y_{\text{new}})$
        $h_{\text{best}} \leftarrow h_{\text{new}}$
      end if
    else
      以概率接受较差解（避免陷入局部最优）：
      $p \leftarrow \exp(-\Delta h / T)$
      if $\text{Random}(0, 1) < p$ **then**
        $(x_{\text{current}}, y_{\text{current}}) \leftarrow (x_{\text{new}}, y_{\text{new}})$
        $h_{\text{current}} \leftarrow h_{\text{new}}$
      end if
    end if
  end for
  降温过程：
  $T \leftarrow \alpha \times T$
end while
return $(x_{\text{best}}, y_{\text{best}}), h_{\text{best}}$

2.2 基于高斯过程的贝叶斯优化（Bayesian Optimization with Gaussian Process）

贝叶斯优化是一种基于概率模型的序列优化策略，特别适用于目标函数评估代价高昂的黑箱优化问题。该方法使用高斯过程作为代理模型来逼近未知的目标函数，并通过采集函数（Acquisition Function）平衡探索和利用，指导下一个评估点的选择。

Algorithm 2 基于高斯过程的贝叶斯优化用于地形最低点搜索

Require: 地图范围 $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$，初始采样点数 n_{init}，总评估预算 N，高斯过程核函数 K（通常使用 Matern 或 RBF 核），采集函数类型（如 EI, PI, UCB）
Ensure: 找到的最低点位置 $(x_{\text{best}}, y_{\text{best}})$ 及其海拔 h_{best}

初始化：
$\mathcal{D} \leftarrow \emptyset$ ▷ 已观测数据集合
初始空间填充设计（如拉丁超立方采样）：
for $i = 1$ to n_{init} **do**
    $(x_i, y_i) \leftarrow \text{LatinHypercubeSample}(\text{map\_range})$
    $h_i \leftarrow \text{GetElevation}(x_i, y_i)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{((x_i, y_i), h_i)\}$
end for
主优化循环：
for $t = n_{\text{init}} + 1$ to N **do**
    基于现有数据 $\mathcal{D}$ 拟合高斯过程模型：
    $\text{GP} \leftarrow \text{FitGaussianProcess}(\mathcal{D}, \text{kernel} = K)$
    根据采集函数选择下一个评估点：
    $(x_{\text{next}}, y_{\text{next}}) \leftarrow \text{OptimizeAcquisitionFunction}(\text{GP}, \mathcal{D}, \text{map\_range})$
    评估新点：
    $h_{\text{next}} \leftarrow \text{GetElevation}(x_{\text{next}}, y_{\text{next}})$
    更新数据集：
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{((x_{\text{next}}, y_{\text{next}}), h_{\text{next}})\}$
    更新当前最优解：
    $(x_{\text{best}}, y_{\text{best}}) \leftarrow \arg\min_{(x,y)\in\mathcal{D}} h(x, y)$
    $h_{\text{best}} \leftarrow \min_{h\in\mathcal{D}} h$
end for
return $(x_{\text{best}}, y_{\text{best}}), h_{\text{best}}$

Algorithm 3 采集函数优化子过程（以期望提升 EI 为例）

function OptimizeAcquisitionFunction(GP, $\mathcal{D}$, map_range)
    计算当前最优值：
    $h_{\text{best}} \leftarrow \min_{h \in \mathcal{D}} h$
    定义期望提升函数：
    function EI(x, y)
        $\mu, \sigma \leftarrow \text{GP.predict}((x, y))$ ▷ 预测均值和标准差
        标准化改进量：
        if $\sigma > 0$ **then**
            $z \leftarrow (h_{\text{best}} - \mu)/\sigma$
            $\text{ei} \leftarrow (h_{\text{best}} - \mu) \cdot \Phi(z) + \sigma \cdot \varphi(z)$
            ▷ 其中 Φ 为标准正态分布 CDF，φ 为 PDF
        else
            $\text{ei} \leftarrow 0$
        end if
        return ei
    end function
    优化 EI 函数以找到下一个评估点：
    $(x_{\text{next}}, y_{\text{next}}) \leftarrow \text{GlobalOptimizer}(\text{EI}, \text{map\_range})$
    return $(x_{\text{next}}, y_{\text{next}})$
end function

2.3 基于插值模型的序列决策（Interpolation-based Optimization, IO）

基于插值模型的序列决策方法通过构建目标函数的插值模型（如径向基函数、Kriging 或多项式模型）来近似未知地形，并基于模型预测和不确定性估计指导搜索过程。与高斯过程不同，这些方法通常使用确定性插值技术，计算效率较高，特别适用于中等维度的连续优化问题。

Algorithm 4 基于径向基函数插值的序列决策优化

Require: 地图范围 $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$，初始采样点数 n_{init}，总评估预算 N，径向基函数类型（如高斯、多二次、逆多二次），权衡参数 λ（平衡模型预测值和不确定性）
Ensure: 找到的最低点位置 $(x_{\text{best}}, y_{\text{best}})$ 及其海拔 h_{best}

初始化：
$X \leftarrow []$ ▷ 已采样点坐标列表
$H \leftarrow []$ ▷ 对应海拔值列表
初始采样（空间填充设计）：
for $i = 1$ to n_{init} **do**
    $(x_i, y_i) \leftarrow \text{SobolSequenceSample}(\text{map\_range}, i)$
    $h_i \leftarrow \text{GetElevation}(x_i, y_i)$
    $X.\text{append}((x_i, y_i))$
    $H.\text{append}(h_i)$
end for
主优化循环：
for $t = n_{\text{init}} + 1$ to N **do**
    构建径向基函数插值模型：
    $\text{RBF\_model} \leftarrow \text{FitRBFInterpolation}(X, H)$
    定义改进函数（平衡预测值和探索性）：
    function ImprovementScore(x, y)
        获取插值预测值（确定性分量）：
        $h_{\text{pred}} \leftarrow \text{RBF\_model.predict}((x, y))$
        计算与最近采样点的距离（不确定性度量）：
        $d_{\min} \leftarrow \min_{(x_i, y_i) \in X} \text{distance}((x, y), (x_i, y_i))$
        综合改进得分（寻找最小值，所以预测值越小越好）：
            ▷ λ 控制探索-利用权衡：$\lambda = 0$ 纯利用，$\lambda > 0$ 增加探索
        $\text{score} \leftarrow h_{\text{pred}} - \lambda \times d_{\min}$
        return score
    end function
    全局优化改进函数：
    $\text{candidates} \leftarrow \text{GenerateCandidatePoints}(\text{map\_range}, X)$
    $\text{scores} \leftarrow [\text{ImprovementScore}(p) \text{ for } p \text{ in candidates}]$
    $(x_{\text{next}}, y_{\text{next}}) \leftarrow \text{candidates}[\arg\min(\text{scores})]$
    评估新点：
    $h_{\text{next}} \leftarrow \text{GetElevation}(x_{\text{next}}, y_{\text{next}})$
    更新数据集：
    $X.\text{append}((x_{\text{next}}, y_{\text{next}}))$
    $H.\text{append}(h_{\text{next}})$
    更新当前最优解：
    $\text{best\_idx} \leftarrow \arg\min(H)$
    $(x_{\text{best}}, y_{\text{best}}) \leftarrow X[\text{best\_idx}]$
    $h_{\text{best}} \leftarrow H[\text{best\_idx}]$
end for
return $(x_{\text{best}}, y_{\text{best}}), h_{\text{best}}$

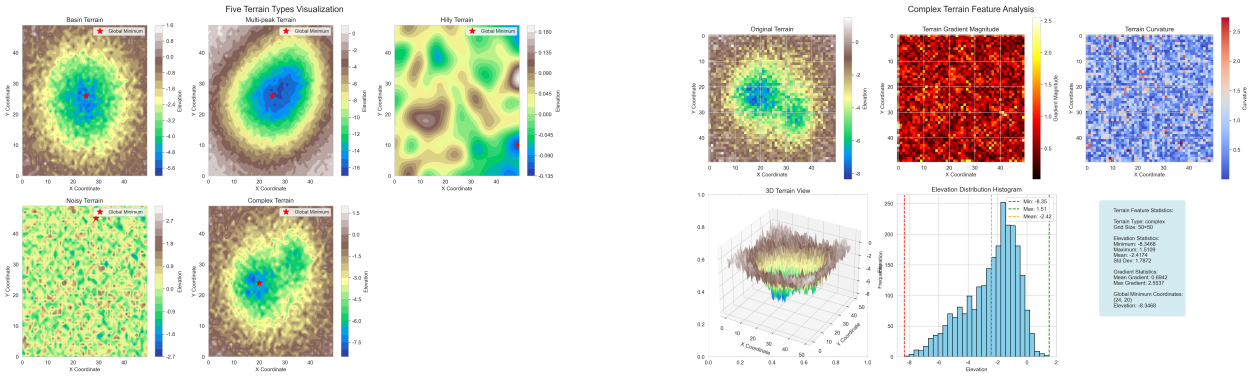
Algorithm 5 候选点生成策略

function GenerateCandidatePoints(map_range, existing_points)
    策略 1：在整个地图范围生成均匀网格点：
    grid_points ← GenerateUniformGrid(map_range, resolution = 50 × 50)
    策略 2：在已有点周围局部搜索：
    local_points ← ∅
    for each (x_i, y_i) in existing_points **do**
                                                ▷ 在每个已有点周围生成随机扰动点
        for $j = 1$ to 5 **do**
            $dx, dy \leftarrow \text{RandomNormal}(0, 0.1 \times \text{map\_range})$
            local_points ← local_points ∪ $\{(x_i + dx, y_i + dy)\}$
        end for
    end for
    合并候选点并确保在地图范围内：
    all_candidates ← grid_points ∪ local_points
    all_candidates ← FilterByBoundary(all_candidates, map_range)
    return all_candidates
end function

3 实验结果数据分析和可视化分析

3.1 地形可视化结果

通过地形生成与可视化模块，我们得到了五种地形的详细可视化结果，如图1所示：



(a) 五种地形类型对比　　(b) 复杂地形 3D 视图

图 1: 地形可视化结果

地形可视化分析揭示了不同地形的结构特征：

- **单盆谷地：**具有明显的单一全局最低点，地形平滑

- **多峰山地**：包含多个局部最低点，搜索空间复杂
- **平缓丘陵**：地形起伏小，梯度变化平缓
- **含噪声地形**：在平缓地形上叠加高频噪声
- **复杂地形**：综合了盆地、山峰和噪声的复杂结构

3.2 地形搜索算法对比实验

3.2.1 实验信息

- 实验时间：2025-12-18 15:18:54
- 地形尺寸：50×50
- 每地形运行次数：10
- 每次运行最大迭代次数：100

3.2.2 实验 1 结果摘要（见表格）

表 1: 实验 1 结果汇总

地形类型	算法	平均误差	误差标准差	平均访问点比率 (%)	成功率 (%)	平均运行时间
盆地 Basin	SA	2.330900	1.420402	2.720	0.0	0.001415
	BO	0.000000	0.000000	4.200	100.0	28.609128
	SDIM	1.090067	0.705591	4.200	0.0	33.161711
多峰 Mult-peak	SA	2.776910	3.696532	2.624	20.0	0.001399
	BO	0.000000	0.000000	4.200	100.0	30.590563
	SDIM	1.489011	0.898011	4.200	0.0	37.477753
丘陵 Hilly	SA	0.062466	0.037460	3.236	0.0	0.001299
	BO	0.006948	0.006242	4.196	60.0	28.821279
	SDIM	0.020632	0.019514	4.200	40.0	33.166224
噪声 Noisy	SA	0.906044	0.493085	2.536	10.0	0.001308
	BO	0.000000	0.000000	4.200	100.0	29.631550
	SDIM	0.853913	0.511597	4.200	10.0	35.314324
复合 Complex	SA	2.251582	1.519006	2.580	0.0	0.001274
	BO	0.000000	0.000000	4.200	100.0	30.071376
	SDIM	1.747507	1.202608	4.200	0.0	34.629563

3.2.3 实验 2 结果摘要

在这个情况下，为了用模拟退火算法得出最好的结果，我们得出的数据是：

- **最佳参数组合**：初始温度 $T_0 = 30$，冷却率 $\alpha = 0.95$
- **使用最佳参数的平均误差**：1.1671
- **使用最佳参数的访问点比率**：2.29%

3.2.4 总体性能比较

从表可以看出，三种算法在不同地形上表现出显著差异：

- **贝叶斯优化 (BO)** 在精度和成功率方面表现最优，在盆地、多峰、噪声和复合四种地形中平均误差为 0，成功率均达到 100%。唯一的例外是在丘陵地形中，其成功率为 60%，平均误差为 0.006948。然而，BO 的计算成本最高，平均运行时间在 28.6-30.6 秒之间，显著长于其他两种算法。
- **模拟退火 (SA)** 在运行时间方面具有绝对优势，平均运行时间仅约 0.0013 秒，比其他两种算法快 4 个数量级。然而，SA 在精度和成功率方面普遍较差，尤其在盆地、丘陵和复合地形中成功率为 0%，在多峰和噪声地形中成功率也仅为 20% 和 10%。
- **序列决策插值模型 (SDIM)** 在性能上介于 BO 和 SA 之间。其平均误差普遍大于 BO 但小于 SA，在丘陵地形中表现最好（平均误差 0.020632，成功率 40%）。SDIM 的访问点比率最高（约 4.2%），运行时间与 BO 相当。

3.2.5 地形对算法性能的影响

不同地形特征对算法性能产生显著影响：

- 在**盆地和平坦地形**中，BO 完美解决了优化问题，而 SA 和 SDIM 均无法找到最低点（成功率为 0%）。这表明对于这种地形，需要更系统性的搜索策略。
- 在**多峰和复合地形**中，BO 再次表现出色，而 SA 和 SDIM 误差较大且成功率低。这表明复杂地形对局部搜索算法（如 SA）和基于插值的方法（如 SDIM）构成了挑战。
- 在**丘陵地形**中，三种算法的性能均有下降。即使是表现最好的 BO，其成功率也仅为 60%。这可能是因为平缓但连续变化的地形难以通过有限的采样点准确建模。
- 在**噪声地形**中，BO 表现出强大的鲁棒性，完全不受噪声影响。相比之下，SA 和 SDIM 受噪声干扰较大，说明这些方法对地形中的随机波动较为敏感。

3.2.6 算法效率分析

从计算效率角度看：

- BO 和 SDIM 访问了约 4.2% 的地图点，而 SA 仅访问了 2.5-3.2% 的点。这表明 SA 采用更激进的搜索策略，但这也导致了较低的精度和成功率。
- BO 的高精度是以高计算成本为代价的，其运行时间比 SA 长约 4 个数量级。这在实时应用场景中可能是不可接受的。
- 误差标准差反映了算法的稳定性。BO 在大多数地形中标准差为 0，表明其具有极好的可重复性。SA 在多峰地形中的标准差高达 3.6965，表明其搜索结果波动较大，不够稳定。

4 从实验结果的角度对算法特征的分析

4.1 模拟退火 SA

我们注意到模拟退火算法存在如下特征：

1. 计算速度特别快：SA 的时间复杂度为 $O(k)$，其中 k 是迭代次数。这意味着 SA 不需要维护复杂的数学模型或进行大量的数学计算，因此在运行时间上远远优与其他两个算法；
2. 访问点比率非常低：SA 仅通过局部信息指导搜索方向。这种缺乏全局视野的搜索策略容易在局部区域重复探索，导致访问点比率较低；
3. 成功率非常小：SA 非常容易陷入局部最优解。虽然概率性接受机制允许算法跳出局部最优，但在复杂地形中成功概率极低。缺乏全局建模能力的算法无法识别远处可能存在的更低点。

4.2 贝叶斯优化算法 BO

作为本场挑战的冠军，BO 存在如下特点：

1. 在盆地和多峰地形中达到 100 % 成功率，但在在丘陵地形中成功率降至 60%：这跟 BO 的具体算法过程有关。高斯过程构建了整个地图的概率模型，通过采集函数识别最有希望的采样点。但是丘陵过程中存在大量的局部最优点，高斯过程的平滑假设可能过度平滑了细微变化，导致错过全局最优；

2. 巨大的计算时间：BO 的核心是高斯过程，其时间复杂度 $O(n^3)$，其中 n 为采样点的数量，这导致 BO 的耗时将会随着采样点的增加急剧上升。同时核函数参数的优化需要多次迭代，这也导致了其所需要的计算时间比另外两个算法都要大上不少；

3. 面对噪声具有较强的对抗能力：贝叶斯优化对噪声的强鲁棒性源于高斯过程的显式噪声建模能力。核函数中包含噪声项：

$$k(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{\|x_i - x_j\|^2}{2l^2}\right) + \sigma_n^2 \delta_{ij}$$

其中 σ_n^2 项显式建模观测噪声。即使存在噪声，高斯过程也能通过平滑效应识别出真实的底层趋势。

4.3 序列决策差值模型 SDIM

SDIM 的正确性介于 SA 和 BO 之间：

1. 精度既不高也不差：大多数插值方法假设目标函数是平滑的，在数据稀疏区域预测不可靠同时特别容易过拟合到噪声数据导致结果出现偏差；

2. 在丘陵地形相对较好，在多峰/噪声地形表现差：插值方法适合平滑连续函数，能够捕捉地形的大致趋势。然而插值可能过度平滑多个峰值，或放大噪声形成虚假低点。缺乏概率框架使其无法区分模型不确定性和观测噪声。

4.4 对地形特征的适应性

三种算法对不同地形特征的适应性总结如表2所示。

表 2: 不同算法对地形特征的适应性

算法	简单地形	多峰地形	噪声地形	混合地形
SA	偶然成功	容易陷入局部最优	随机干扰搜索	表现最差
BO	完美解决	智能平衡多个最优	显式建模噪声	稳健最优
SDIM	中等表现	过平滑失去细节	噪声误导插值	表现不稳定

5 实用建议与未来工作

基于上述实验结果，提出以下建议：

1. 在对精度和成功率要求较高的场景中，应优先选择贝叶斯优化，尤其是在计算资源充足、时间限制不严格的情况下。

2. 在实时或低延迟应用中，模拟退火是更合适的选择，尽管需要接受较低的精度和成功率。

3. 序列决策插值模型在当前测试中未显示出明显优势，除非问题结构比较平滑或者满足其他特殊性质使得其特别适合其插值假设。

4. 未来研究可考虑混合策略：结合 SA 的快速初探和 BO 的精细搜索，或开发自适应采样策略以平衡探索与利用。此外，并行计算技术可用于加速 BO 的计算过程。

综上所述，贝叶斯优化在寻找最低海拔点的任务中表现出最佳的整体性能，但其高计算成本限制了在实时场景中的应用。模拟退火虽然速度快，但精度有限，适用于对实时性要求极高的场景。地形特征对算法性能有显著影响，在实际应用中应考虑结合地形先验信息选择合适的优化策略。

6 参考文献

参考文献

[1] C. Shi, X. Yuan, and Z. Jiang, "Gps elevation fitting of bp neural network optimized by genetic simulated annealing algorithm," *Global Positioning System*, no. 5, 2021.

[2] B. Lin and Z. Chen, "An improvement of hill climbing algorithm with jumping strategy," *Journal of National Quemoy University*, vol. 4, pp. 105–116, 2010.

[3] F. Zhang, J. Song, J. C. Bowden, A. Ladd, Y. Yue, T. Desautels, and Y. Chen, "Learning regions of interest for bayesian optimization with adaptive level-set estimation," in *Proceedings of the 40th International Conference on Machine Learning*, vol. 202 of *Proceedings of Machine Learning Research*, pp. 41579–41595, 2023.

[4] F. Jimenez and M. Katzfuss, "Scalable bayesian optimization using vecchia approximations of gaussian processes," *arXiv preprint arXiv:2203.01459*, 2022.

[5] L. Liu *et al.*, "Influence of terrain factors on optimal order distance of inverse distance weighted (idw)," *Scientia Geographica Sinica*, vol. 43, no. 7, pp. 1281–1290, 2023.

A 附录

本论文的所有数据和代码可以在这个 Github 仓库中查询到。以下列出部分重要代码。

A.1 实验代码实现

本附录提供了本文实验中使用的三种算法的核心代码实现。

A.1.1 模拟退火算法实现

模拟退火算法的核心代码如代码1所示。该算法通过在解空间中随机游走，以一定概率接受较差的解，从而避免陷入局部最优。

```
class SimulatedAnnealing:
    """Simulated Annealing Algorithm"""

    def __init__(self, terrain, T0=50, cooling_rate=0.95, max_iter=200,
        adaptive_T0=False):
        self.terrain = terrain
        self.grid_size = terrain.shape[0]
        self.T0 = T0
        self.cooling_rate = cooling_rate
        self.max_iter = max_iter
        self.adaptive_T0 = adaptive_T0

    def search(self):
        """Execute simulated annealing search"""
        # Random starting point
        current = (np.random.randint(self.grid_size), np.random.randint(
            self.grid_size))
        best = current
        H_current = self.terrain[current]
        H_best = H_current

        # Adaptive initial temperature calculation
        T = self.T0
        if self.adaptive_T0:
            samples = []
            for _ in range(100):
                move = np.random.randint(-5, 6, size=2)
                neighbor = ((current[0] + move[0]) % self.grid_size,
```

```
                            (current[1] + move[1]) % self.grid_size)
                samples.append(abs(self.terrain[neighbor] - H_current))
            T = 5 * np.std(samples)
            if T < 1:
                T = 10

        # Initialize records
        history = []
        visited = set([current])

        # Simulated annealing search
        for i in range(self.max_iter):
            # Generate neighborhood random move
            move = np.random.randint(-2, 3, size=2)
            neighbor = ((current[0] + move[0]) % self.grid_size,
                        (current[1] + move[1]) % self.grid_size)

            H_neighbor = self.terrain[neighbor]
            delta_H = H_neighbor - H_current

            # Metropolis criterion
            if delta_H < 0 or np.random.rand() < np.exp(-delta_H / (T + 1e
                -10)):
                current = neighbor
                H_current = H_neighbor
                visited.add(current)

                if H_current < H_best:
                    best = current
                    H_best = H_current

            history.append(H_best)
            T *= self.cooling_rate

            # Early termination condition
            if T < 1e-5:
                break

        return best, H_best, history, visited
```

Listing 1: 模拟退火算法核心代码

A.1.2 贝叶斯优化算法实现

贝叶斯优化算法的核心代码如代码2所示。该算法通过高斯过程建模目标函数，并使用采集函数指导下一步采样点。

```
class BayesianOptimization:
    """Bayesian Optimization Algorithm based on Gaussian Process"""

    def __init__(self, terrain, n_initial=5, max_iter=200,
        exploration_weight=0.1):
        self.terrain = terrain
        self.grid_size = terrain.shape[0]
        self.n_initial = n_initial
        self.max_iter = max_iter
        self.exploration_weight = exploration_weight

    def search(self):
        """Execute Bayesian optimization search"""
        # Initialize dataset
        X = []
        y = []
        visited = set()

        # Initial random sampling
        for _ in range(self.n_initial):
            point = (np.random.randint(self.grid_size), np.random.randint(
                self.grid_size))
            value = self.terrain[point]
            X.append([point[0], point[1]])
            y.append(value)
            visited.add(point)

        # Record history
        history = [min(y)] if y else []

        # Bayesian optimization loop
        for iteration in range(self.max_iter):
            # Simplified version: use nearest neighbor as surrogate model
            # In real applications, Gaussian Process Regression should be
                used here

            # Generate candidate point grid
```

```
            candidates = []
            for i in range(self.grid_size):
                for j in range(self.grid_size):
                    candidates.append((i, j))

            # Calculate score for each candidate (simplified version)
            scores = []
            for candidate in candidates:
                if candidate in visited:
                    scores.append(-float('inf'))
                    continue

                # Calculate minimum distance to sampled points
                min_dist = float('inf')
                for xi in X:
                    dist = np.sqrt((candidate[0]-xi[0])**2 + (candidate[1]-
                        xi[1])**2)
                    min_dist = min(min_dist, dist)

                # Simplified acquisition function: balance exploration and
                    exploitation
                score = -self.terrain[candidate] + self.exploration_weight
                    * min_dist
                scores.append(score)

            # Select next sampling point
            if scores:
                next_idx = np.argmax(scores)
                next_point = candidates[next_idx]
            else:
                next_point = (np.random.randint(self.grid_size), np.random.
                    randint(self.grid_size))

            # Sample and update dataset
            value = self.terrain[next_point]
            X.append([next_point[0], next_point[1]])
            y.append(value)
            visited.add(next_point)

            # Update history best value
            history.append(min(y))
```

```
72
73          # Return best point
74          best_idx = np.argmin(y)
75          best_point = (int(X[best_idx][0]), int(X[best_idx][1]))
76          best_value = y[best_idx]
77
78          return best_point, best_value, history, visited
```

Listing 2: 贝叶斯优化算核心代码

A.1.3 序列决策插值模型实现

序列决策插值模型的核心代码如代码3所示。该算法通过插值方法预测未探索区域，并选择最可能包含最低点的区域进行探索。

```
1  class SequentialDecision:
2      """Sequential Decision Algorithm based on Interpolation Model"""
3
4      def __init__(self, terrain, n_initial=5, max_iter=200,
           uncertainty_weight=0.2):
5          self.terrain = terrain
6          self.grid_size = terrain.shape[0]
7          self.n_initial = n_initial
8          self.max_iter = max_iter
9          self.uncertainty_weight = uncertainty_weight
10
11     def search(self):
12         """Execute sequential decision search"""
13         # Initialize dataset
14         X = []
15         y = []
16         visited = set()
17
18         # Initial random sampling
19         for _ in range(self.n_initial):
20             point = (np.random.randint(self.grid_size), np.random.randint(
                   self.grid_size))
21             value = self.terrain[point]
22             X.append([point[0], point[1]])
23             y.append(value)
24             visited.add(point)
25
```

```
        # Record history
        history = [min(y)] if y else []

        # Sequential decision loop
        for iteration in range(self.max_iter):
            if len(X) < 3:
                # Too few samples, random sampling
                next_point = (np.random.randint(self.grid_size), np.random.randint(self.grid_size))
            else:
                try:
                    # Build interpolation model
                    X_array = np.array(X)
                    y_array = np.array(y)

                    # Use radial basis function interpolation
                    rbf = Rbf(X_array[:, 0], X_array[:, 1], y_array, function='gaussian')

                    # Generate prediction grid
                    grid_x, grid_y = np.meshgrid(np.arange(self.grid_size), np.arange(self.grid_size))

                    # Predict terrain
                    Z_pred = rbf(grid_x, grid_y)

                    # Calculate uncertainty (based on distance)
                    uncertainty = np.zeros_like(Z_pred)
                    for i in range(self.grid_size):
                        for j in range(self.grid_size):
                            # Calculate minimum distance to all sampled points
                            min_dist = float('inf')
                            for xi in X:
                                dist = np.sqrt((i-xi[0])**2 + (j-xi[1])**2)
                                min_dist = min(min_dist, dist)
                            uncertainty[i, j] = min_dist

                    # Normalize uncertainty
                    if uncertainty.max() > 0:
                        uncertainty = uncertainty / uncertainty.max()
```

```
63
64                      # Select next sampling point (balance prediction and
                            uncertainty)
65                      scores = -Z_pred + self.uncertainty_weight *
                            uncertainty
66
67                      # Exclude visited points
68                      for i in range(self.grid_size):
69                          for j in range(self.grid_size):
70                              if (i, j) in visited:
71                                  scores[i, j] = -float('inf')
72
73                      # Select point with highest score
74                      next_idx = np.unravel_index(np.argmax(scores), scores.
                            shape)
75                      next_point = (next_idx[0], next_idx[1])
76
77                  except Exception as e:
78                      # If interpolation fails, random sampling
79                      next_point = (np.random.randint(self.grid_size), np.
                            random.randint(self.grid_size))
80
81              # Sample and update dataset
82              value = self.terrain[next_point]
83              X.append([next_point[0], next_point[1]])
84              y.append(value)
85              visited.add(next_point)
86
87              # Update history best value
88              history.append(min(y))
89
90          # Return best point
91          best_idx = np.argmin(y)
92          best_point = (int(X[best_idx][0]), int(X[best_idx][1]))
93          best_value = y[best_idx]
94
95          return best_point, best_value, history, visited
```

Listing 3: 序列决策插值模型核心代码

A.1.4 地形生成代码

为了生成本文中的实验数据，我们创建了多种地形生成函数。关键的地形生成代码如代码4所示。

```
class TerrainGenerator:
    """Terrain Generator Class"""

    def __init__(self, grid_size=50):
        self.grid_size = grid_size
        self.terrain_types = ['basin', 'multi-peak', 'hilly', 'noisy', '
            complex']

    def generate_terrain(self, terrain_type='basin', seed=42):
        """Generate specified terrain type"""
        np.random.seed(seed)
        terrain = np.random.randn(self.grid_size, self.grid_size) * 0.5

        if terrain_type == 'basin':
            # Single basin terrain
            x = np.linspace(-2, 2, self.grid_size)
            y = np.linspace(-2, 2, self.grid_size)
            X, Y = np.meshgrid(x, y)
            basin = -np.exp(-(X**2 + Y**2)/2) * 5
            terrain += basin

        elif terrain_type == 'multi-peak':
            # Multi-peak terrain
            for _ in range(8):
                cx = np.random.uniform(0.2, 0.8) * self.grid_size
                cy = np.random.uniform(0.2, 0.8) * self.grid_size
                sx = np.random.uniform(5, 15)
                sy = np.random.uniform(5, 15)
                depth = np.random.uniform(2, 5)

                X, Y = np.meshgrid(np.arange(self.grid_size), np.arange(
                    self.grid_size))
                gaussian = -np.exp(-((X-cx)**2/(2*sx**2) + (Y-cy)**2/(2*sy
                    **2))) * depth
                terrain += gaussian

        elif terrain_type == 'hilly':
```

```
            # Hilly terrain
            terrain = ndimage.gaussian_filter(terrain, sigma=3)

        elif terrain_type == 'noisy':
            # Noisy terrain
            terrain = ndimage.gaussian_filter(terrain, sigma=2)
            terrain += np.random.randn(self.grid_size, self.grid_size) *
                0.8

        elif terrain_type == 'complex':
            # Complex terrain
            x = np.linspace(-2, 2, self.grid_size)
            y = np.linspace(-2, 2, self.grid_size)
            X, Y = np.meshgrid(x, y)
            basin = -np.exp(-(X**2 + Y**2)/3) * 4
            terrain += basin

            for _ in range(5):
                cx = np.random.uniform(0.2, 0.8) * self.grid_size
                cy = np.random.uniform(0.2, 0.8) * self.grid_size
                sx = np.random.uniform(3, 8)
                sy = np.random.uniform(3, 8)
                depth = np.random.uniform(1, 3)

                X, Y = np.meshgrid(np.arange(self.grid_size), np.arange(
                    self.grid_size))
                gaussian = -np.exp(-((X-cx)**2/(2*sx**2) + (Y-cy)**2/(2*sy
                    **2))) * depth
                terrain += gaussian * 0.7

            terrain += np.random.randn(self.grid_size, self.grid_size) *
                0.5

        return terrain

    def visualize_all_terrains(self, save_fig=True):
        """Visualize all terrain types"""
        print("Generating and visualizing five terrain types...")

        fig, axes = plt.subplots(2, 3, figsize=(15, 10))
        axes = axes.flatten()
```

```
72
73          # Terrain name mapping
74          terrain_name_map = {
75              'basin': 'Basin Terrain',
76              'multi-peak': 'Multi-peak Terrain',
77              'hilly': 'Hilly Terrain',
78              'noisy': 'Noisy Terrain',
79              'complex': 'Complex Terrain'
80          }
81
82          for i, terrain_type in enumerate(tqdm(self.terrain_types, desc="
                Generating terrains")):
83              terrain = self.generate_terrain(terrain_type)
84
85              ax = axes[i]
86              contour = ax.contourf(terrain, levels=20, cmap='terrain')
87
88              # Use English labels
89              english_name = terrain_name_map.get(terrain_type, terrain_type)
90              ax.set_title(f'{english_name}', fontsize=12)
91              ax.set_xlabel('X Coordinate')
92              ax.set_ylabel('Y Coordinate')
93
94              # Mark the global minimum
95              min_pos = np.unravel_index(np.argmin(terrain), terrain.shape)
96              ax.plot(min_pos[1], min_pos[0], 'r*', markersize=12, label='
                    Global Minimum')
97              ax.legend(fontsize=9)
98
99              plt.colorbar(contour, ax=ax, label='Elevation')
100
101         axes[-1].axis('off')
102         plt.suptitle('Five Terrain Types Visualization', fontsize=16, y
                =0.98)
103         plt.tight_layout()
104
105         if save_fig:
106             plt.savefig('1_Terrain_Types.png', dpi=300, bbox_inches='tight'
                    )
107             print("Terrain visualization saved as '1_Terrain_Types.png'")
108
```

```
        plt.show()

        return fig

    def analyze_terrain_features(self, terrain_type='complex', save_fig=
        True):
        """Analyze and visualize terrain features"""
        print(f"Analyzing {terrain_type} terrain features...")

        terrain = self.generate_terrain(terrain_type)

        # Calculate global minimum position
        global_min_pos = np.unravel_index(np.argmin(terrain), terrain.shape
            )

        fig, axes = plt.subplots(2, 3, figsize=(15, 10))

        # Original terrain
        ax1 = axes[0, 0]
        im1 = ax1.imshow(terrain, cmap='terrain')
        ax1.set_title('Original Terrain')
        ax1.set_xlabel('X Coordinate')
        ax1.set_ylabel('Y Coordinate')
        plt.colorbar(im1, ax=ax1, label='Elevation')

        # Terrain gradient
        ax2 = axes[0, 1]
        grad_x, grad_y = np.gradient(terrain)
        gradient_magnitude = np.sqrt(grad_x**2 + grad_y**2)
        im2 = ax2.imshow(gradient_magnitude, cmap='hot')
        ax2.set_title('Terrain Gradient Magnitude')
        ax2.set_xlabel('X Coordinate')
        ax2.set_ylabel('Y Coordinate')
        plt.colorbar(im2, ax=ax2, label='Gradient Magnitude')

        # Terrain curvature
        ax3 = axes[0, 2]
        smoothed = ndimage.gaussian_filter(terrain, sigma=1)
        grad_xx, grad_xy = np.gradient(grad_x)
        grad_yx, grad_yy = np.gradient(grad_y)
        curvature = np.abs(grad_xx + grad_yy)
```

```
        im3 = ax3.imshow(curvature, cmap='coolwarm')
        ax3.set_title('Terrain Curvature')
        ax3.set_xlabel('X Coordinate')
        ax3.set_ylabel('Y Coordinate')
        plt.colorbar(im3, ax=ax3, label='Curvature')

        # 3D terrain view
        ax4 = fig.add_subplot(2, 3, 4, projection='3d')
        X, Y = np.meshgrid(np.arange(self.grid_size), np.arange(self.
            grid_size))
        surf = ax4.plot_surface(X, Y, terrain, cmap='terrain', alpha=0.8,
            linewidth=0)
        ax4.set_title('3D Terrain View')
        ax4.set_xlabel('X Coordinate')
        ax4.set_ylabel('Y Coordinate')
        ax4.set_zlabel('Elevation')

        # Elevation distribution histogram
        ax5 = axes[1, 1]
        ax5.hist(terrain.flatten(), bins=30, color='skyblue', edgecolor='
            black')
        ax5.set_title('Elevation Distribution Histogram')
        ax5.set_xlabel('Elevation')
        ax5.set_ylabel('Frequency')
        ax5.axvline(x=terrain.min(), color='red', linestyle='--',
                    label=f'Min: {terrain.min():.2f}')
        ax5.axvline(x=terrain.max(), color='green', linestyle='--',
                    label=f'Max: {terrain.max():.2f}')
        ax5.axvline(x=terrain.mean(), color='orange', linestyle='--',
                    label=f'Mean: {terrain.mean():.2f}')
        ax5.legend()

        # Terrain statistics
        ax6 = axes[1, 2]
        ax6.axis('off')

        stats_text = f"""
        Terrain Feature Statistics:

        Terrain Type: {terrain_type}
        Grid Size: {self.grid_size}×{self.grid_size}
```

```
186
187         Elevation Statistics:
188         Minimum: {terrain.min():.4f}
189         Maximum: {terrain.max():.4f}
190         Mean: {terrain.mean():.4f}
191         Std Dev: {terrain.std():.4f}
192
193         Gradient Statistics:
194         Mean Gradient: {gradient_magnitude.mean():.4f}
195         Max Gradient: {gradient_magnitude.max():.4f}
196
197         Global Minimum Coordinates:
198         ({global_min_pos[0]}, {global_min_pos[1]})
199         Elevation: {terrain[global_min_pos]:.4f}
200         """
201
202         ax6.text(0.1, 0.95, stats_text, transform=ax6.transAxes,
203                 fontsize=10, verticalalignment='top',
204                 bbox=dict(boxstyle="round,pad=0.5", facecolor="lightblue",
                        alpha=0.5))
205
206         plt.suptitle(f'{terrain_type.capitalize()} Terrain Feature Analysis
                ', fontsize=16, y=0.98)
207         plt.tight_layout()
208
209         if save_fig:
210             plt.savefig(f'2_{terrain_type}_Terrain_Analysis.png', dpi=300,
                    bbox_inches='tight')
211             print(f"Terrain analysis saved as '2_{terrain_type}
                    _Terrain_Analysis.png'")
212
213         plt.show()
214
215         return fig, terrain
```

Listing 4: 地形数据生成代码