arXiv:2301.08713v1 [cs.NE] 20 Jan 2023

通过种群异步传播实现大规模并行遗传优化

Oskar Taubert[0000−0002−3707−499X]、Marie Weiel[0000−0001−9648−4385]、DanielCoquelin[0000−0001−8552−5153]、Anis Farshian[0000−0002−9888−0653]、CharlotteDebus[0000−0002−7156−2022]、Alexander Schug[0000−0002−0534−502X]、Achim Streit[0000−0002−5065−469X]和Markus G¨otz[0000−0002−2233−1041]

德国卡尔斯鲁厄理工学院（KIT）Steinbuch计算中心（SCC），邮编76344, Eggenstein-Leopoldshafen，markus.goetz@kit.edu

摘要。 我们介绍了 Propulate，这是一种用于全局优化、尤其是超参数搜索的进化优化算法和软件包。为高效利用高性能计算资源，Propulate 不像传统遗传算法那样在每一代之后进行同步。相反，它会根据繁殖新个体时已有的完整种群来引导搜索。我们提供了该算法的消息传递接口（MPI）实现，其中包含多种选择、变异、交叉和迁移方案，并且易于扩展自定义功能。我们将 Propulate 与成熟的优化工具 Optuna 进行了比较。结果表明，Propulate 的速度最高可快三个数量级，同时不牺牲解的准确性，这证明了我们惰性同步方法的效率和效能。代码和文档见 https://github.com/Helmholtz-AI-Energy/propulate。

关键词： 遗传优化 · 人工智能 · 并行化 · 进化算法

1 引言

如今，机器学习（ML）算法已广泛应用于人类生活的几乎所有领域，从医学诊断和关键基础设施，到交通运输和食品生产。几乎所有机器学习算法都具有不可学习的超参数（HP），这些参数会影响训练过程，尤其会影响模型的预测能力。由于评估一组超参数至少需要进行部分训练，网格搜索和随机搜索等无状态超参数优化方法往往会超出可用计算资源的限制 [15]。为高效探索高维超参数空间，必须利用以往评估的信息来引导搜索。这类依赖状态的策略能以尽可能少的评估次数找到实用模型，从而缩短搜索时间，降低计算能耗。贝叶斯优化器和仿生优化器是最受欢迎的自动化机器学习方法。在后者中，遗传算法（GAs）是一种受自然进化启发、用途广泛的元启发式算法。

为解决搜索解的问题，一组候选解（或个体）组成的种群通过选择与变异的迭代交替过程不断进化 [23,30]。虽然遗传算法并不保证找到全局最优解，但与经典优化器相比，它们往往能以较少的计算开销找到近似最优解 [9,8]。遗传算法已广泛应用于各种优化问题，包括机器学习的超参数优化以及神经架构搜索（NAS） [14]。

为了充分利用日益庞大的模型和数据集，为高性能计算（HPC）设计可扩展算法已成为必需之举 [40]。贝叶斯优化本质上是串行的，而遗传算法（GAs）的结构使其适合并行化 [34]：由于每次迭代中的所有候选解彼此独立，因此可以并行评估。然而，要产生下一代，必须先完成上一代。由于评估不同候选解的计算开销各不相同，同步并行进化过程会造成显著瓶颈，从而影响可扩展性。诸如岛屿模型（IM） [34] 这类旨在减少并行遗传算法整体通信量的方法，并未解决根本的同步问题。

为解决显式同步带来的问题，我们推出了 Propulate：一种支持种群异步传播和迁移的大规模并行遗传优化器。与经典遗传算法不同，Propulate 维护一个由已评估个体构成的连续种群，并弱化了通常严格划分的离散代的概念。我们的贡献包括：

– 一种新颖的并行遗传算法，基于完全异步的岛屿模型，各工作进程独立执行，可实现优化过程的并行化，并分布式地进行目标函数的内部评估。– 通过连续种群的异步传播与迁移，实现大规模并行。– 使用 Python 实现的原型，通过消息传递接口（MPI）实现极高效的通信。– 通过最大限度减少高性能计算系统中的空闲时间，充分利用并行硬件。

我们使用 Propulate 在超级计算机上优化多种基准函数以及深度神经网络的超参数。将结果与广受欢迎的超参数优化软件包 Optuna 的结果进行比较后，我们发现，Propulate 的速度始终快得多，同时不会牺牲解的准确性。我们还进一步证明，Propulate 能够良好扩展至至少 100 个处理单元（PE），且效率没有明显损失，从而展示了我们异步进化方法的有效性。

2 相关工作

近年来，机器学习取得的进展推动了这些技术的大量应用，而 Python 已成为事实上的标准编程语言。调优超参数需要解决高维优化问题，其中机器学习算法是黑盒，模型性能指标则充当目标函数（目标函数）。最常见的是贝叶斯优化器（例如 Optuna [2], Hyperopt [7], SMAC3 [24,27], Spearmint [32],GPyOpt [5], 和 MOE [38]）以及受生物启发的方法，例如基于群体的算法（例如

FLAPS [39]）和进化算法（例如 DEAP [16], MENNDL [40]）。下文将概述 Python 中广泛使用的超参数优化器，重点介绍具有状态依赖性的并行算法及其实现。关于并行遗传算法的理论综述，可参见相关综述文章 [12,4,3] 和书籍 [37,29]。

Optuna采用多种超参数采样和剪枝算法，用于剔除前景不佳的试验，包括树结构帕岑估计器（TPE）、高斯过程和协方差矩阵自适应进化策略。它通过关系数据库服务器支持并行运行。在并行情况下，Optuna 候选解会从磁盘获取先前候选解的信息，并将结果存储到磁盘。

SMAC3（**顺**序**模**型**算**法**配**置）将基于随机森林的贝叶斯方法与激进的竞速机制相结合 [24]。其并行变体 pSMAC 使用多个协同运行的 SMAC3 实例，这些实例通过文件系统共享评估结果。

Spearmint、GPyOpt 和 MOE 都是基于高斯过程的贝叶斯优化器。Spearmint 通过 Sun Grid Engine 和 MongoDB 支持分布式超参数优化。GPyOpt 已集成到 Sherpa 软件包中 [22], ，该软件包提供近期超参数优化器的实现，并提供基础设施，使其能够通过网格引擎和数据库服务器并行运行。MOE（**度**量**优**化**引**擎）使用单步贝叶斯最优算法，在并行设置下最大化多点期望改进 [38]。它采用基于 REST 的客户端-服务器模型，通过分发每次评估并同时运行多次评估来实现多级并行性。

Nevergrad [31] 和 Autotune [25] 提供无梯度优化器和进化优化器，包括贝叶斯优化、粒子群优化和单次优化。在 Nevergrad 中，并行评估通过 Python 的 concurrent 模块提供的执行器使用多个工作进程。Autotune 支持全局搜索与局部搜索并发执行、跨方法共享评估结果、方法混合以及多层并行性。Open Source Vizier [33] 是 Google 超参数优化服务 Vizier 的 Python 接口。它实现了高斯过程赌博机 [19] ，并支持动态切换优化器。中央数据库服务器负责算法提案工作，客户端执行评估，并通过远程过程调用与服务器通信。Katib [18] 是一个基于 Kubernetes 容器编排系统的云原生自动化机器学习项目。它与 Optuna 和 Hyperopt 集成。Tune [26] 基于 Ray 分布式计算平台构建。它支持 Optuna、Hyperopt 和 Nevergrad，并利用多层并行性。

DEAP（**D**istributed **E**volutionary **A**lgorithmsin **P**ython） [16] 实现了通用遗传算法、进化策略、多目标优化以及多种群协同进化。它支持通过 Python 的多进程或 SCOOP 模块实现并行化。EvoTorch [36] 基于 PyTorch 构建，实现了基于分布和种群的算法。借助 Ray 集群，它可以扩展到多颗 CPU、GPU 和多台计算机。MENNDL（**M**ulti-node **E**volutionary **N**euralNetworks for **D**eep **L**earning） [40] 是一款闭源、通过 MPI 并行化的高性能计算优化器，用于自动化网络选择。主节点负责遗传操作，其余工作节点负责评估。然而，全局同步妨碍了资源利用率的优化 [40]。

算法 1：基本遗传算法。在每一代中，根据优化问题的目标函数对个体进行评估——
。遗传算子
将它们传播到下一代：选择算子会挑选当前代中的
一部分个体，通常优先选择表现较好的个体。为了繁育新的个体，
两个或多个父代的基因
对选择池中的个体进行操作。当交叉
算子重组父代个体的基因时，变异算子会对
它们进行随机修改。重复此过程，直到满足停止条件。

输入： 搜索空间范围、种群规模 P, *termination condition*, *selection_policy*, *crossover_probability*, *mutation_probability*。

在搜索空间内初始化由 P 个体组成的种群 *pop* 。
当未 *termination condition* **时执行** // 优化
  在 *pop* 中评估个体。 // 评估
  从以下 *selection policy* 中选择 *parents* *pop*。 // 选择
  对 *individual* 中 *pop* **的每个元素执行** // 变异
    如果 *random* $\leq$ *crossover probability* **，则** // 重组
      从 *parents* 中随机选择个体进行重组。
    如果 *random* $\leq$ *mutation probability* **，则** // 变异
      变异。
    更新 *individual* in *pop*。

结果： 找到的最优个体（即最小化时目标函数值最低的个体）。

3 Propulate 算法与实现

为缓解同步并行遗传算法固有的瓶颈，我们的大规模并行遗传优化器 Propulate（***prop****agate* 和 *pop****ulate***）实现了一种专为大规模高性能计算系统设计的完全异步岛屿模型。与传统遗传算法不同，Propulate 维护着一个由已评估个体构成的连续种群，并弱化了通常严格分隔各代的概念。这使得个体的 *asynchronous* 评估、变异、传播和迁移得以实现。

Propulate 的基本机制是达尔文进化，即选择、重组并变异有益性状，以繁育适应性更强的个体（参见算法 1）。在更高层面上，Propulate 采用岛屿模型，将彼此独立演化的自治子种群与间歇性的个体交换结合起来 [34]。为协调全局搜索，每个岛屿会不时将迁移个体送往目标岛屿的种群。岛屿之间通过竞争方式交流遗传信息，因此与泛交模型相比，子种群的多样性更高 [11]。对于同步岛屿模型，这种交换会在固定间隔后同时进行，其间不执行任何计算。以下超参数用于描述岛屿模型：

– 岛屿数量和子种群规模 – 迁移（授粉）概率 – 迁移者数量（授粉者）： 每次从源种群迁移的个体数量。

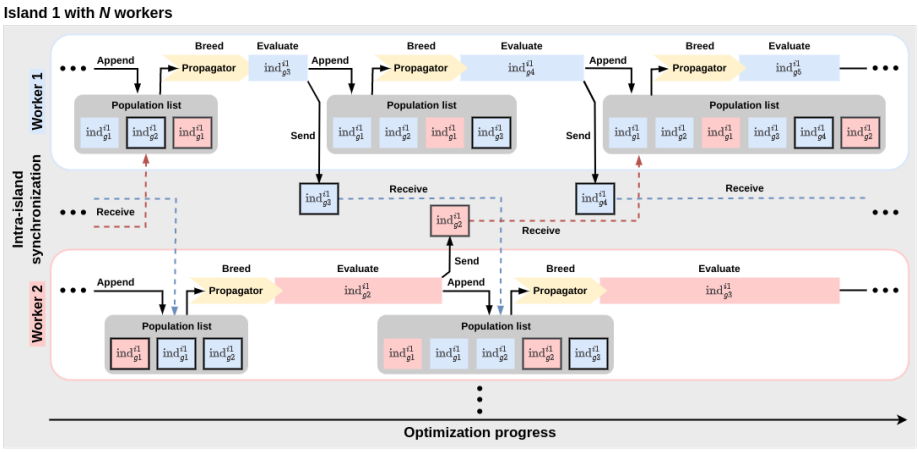

图 1. 异步传播。同一岛屿上两个工作进程之间的交互。工作进程 1 和 2 培育的个体分别以蓝色和红色表示。它们的来源通过代数下标和岛屿上标标示。种群以圆角灰色方框表示，其中最新的个体带有黑色轮廓。不同的评估时间用宽度各异的尖角方框表示。我们以蓝色个体 ind_{g3}^{i1} 为例，说明种群的异步传播和岛内同步。工作进程 1 在第 3 代中培育该个体，方法是将传播器（黄色）应用于工作进程当前的种群。评估完 ind_{g3}^{i1} 后，工作进程 1 将其发送给岛上的所有工作进程，并将其加入自己的种群。由于没有等待接收的、由工作进程 2 发送的已评估个体，工作进程 1 便继续培育个体。工作进程 2 只有在完成对红色个体 ind_{g2}^{i1} 的评估后，才会收到蓝色个体 ind_{g3}^{i1} 。随后，它将这两个个体都加入自己的种群，并培育出一个第 3 代的新个体。

– **迁移（授粉）拓扑：** 岛屿之间迁移（授粉）路径的有向图。– **移出策略：** 如何选择移出个体（例如随机选择或选择最优个体），以及是否将其从源种群中移除（实际迁移）或不移除（授粉）。– **迁入策略：** 如何将移入个体纳入目标种群，即将其添加到种群中（迁移），或替换现有个体（授粉，例如随机替换或替换最差个体）。

Propulate 的工作原理如算法 2 所示。我们考虑将多个处理单元（或工作进程）划分到不同岛屿中。每个工作进程一次处理一个个体，并维护一个种群，用于跟踪其所在岛屿上已评估和迁移的个体。为减少同步目标函数评估带来的计算开销，Propulate 采用连续种群的异步传播方式，并交织使用特定于各工作进程的代（见图 1）。在每次迭代中，每个工作进程都会繁育并评估一个个体，然后将其加入种群列表。随后，它会将该个体及其评估结果发送给该

算法 2：采用授粉的 Propulate。

输入： 搜索空间范围；超参数 *n islands*，岛屿规模 P_i ($i = 1, \ldots, n$ *islands*)，迭代次数 *generations*，进化算子（包括*selection policy*、 *crossover probability*, *mutation probability* 等）、 *pollination probability*、 *pollination_topology*, *emigration_policy*, *immigration_policy*。

配置 *n islands* 个岛屿，每个岛屿配备 P_i 个工作进程。每个工作进程每次评估一个个体，并维护自己的种群列表 *pop* ，其中包含已评估的个体以及迁移到岛上的个体。

/* 开始优化。 */

对每个工作进程**并行执行**

  当 *generation* $\leq$ *generations* **执行**// 遍历各代。

    繁育并评估个体。将其追加到 *pop*。将其发送给其他岛屿上的工作进程，以同步其种群列表：评估个体() // 繁育并评估

    检查并视情况接收由其他工作进程繁育并评估的个体岛屿上的工作进程。将它们附加到 *pop*：接收岛内个体() // 同步

    如果 *random* $\leq$ *pollination probability* **则** // 迁出

      从岛屿上当前活跃的个体中选择授粉者根据 *emigration policy*。向目标岛屿的工作进程发送授粉者副本，依据 *pollination_topology*：发送移出个体()

    检查并视情况接收其他工作进程发送的授粉者，这些工作进程来自其他岛。将它们添加到 *pop*。根据 *pop* 确定要被替换的个体由传入的授粉者替换。发送要停用的、将替换其他岛屿上工作进程的个体：接收移入个体() // 迁移

    检查并视情况接收授粉者在岛屿上的其他工作进程所替换的个体。尝试在 *pop* 中停用它们。如果某个待停用的个体尚未在 *pop* 中，则将其追加到历史列表 *replaced* 并在下一代重试：停用已替换的个体() // 同步

    进入下一代：代 `+= 1`

  /* 优化完成：最终同步 */

  等待所有其他工作进程完成：MPI.COMM WORLD.barrier()

  最后检查是否收到消息，确保所有工作进程都持有完整种群。

  探查由岛上其他工作进程评估的个体：接收岛内个体()

  探查从其他岛迁入的授粉者：接收移入个体()

  探查被岛上其他工作进程替换的个体，以便已停用：停用被替换的个体()

结果： n 目标函数值最小的个体。

同一岛屿的工作进程会接收它们发送的、已经评估的个体，从而相互更新各自的种群列表。为避免显式同步点，独立运行的工作进程通过消息传递接口进行异步点对点通信，以共享结果。每个工作进程在完成一次评估后，立即发送其结果。紧接着，它会以非阻塞方式检查是否有来自同一岛屿工作进程、等待接收的消息。在下一次迭代中，它会对岛屿上各代所有已评估个体构成的连续种群列表应用进化算子，繁衍出一个新个体。因此，尽管个体的计算成本各不相同，工作进程仍能异步运行而不会出现空闲时间。

相互更新后，各岛屿之间会以工作进程为单位，按照一定概率进行异步迁移或授粉。每个工作进程都会从当前种群中选择若干移出个体。对于实际迁移[1]，，一个个体只能在一个岛屿上处于活跃状态。因此，为避免其他工作进程重复选择，工作进程只能从岛屿种群中专属的子集中选择符合条件的移出个体。随后，它会按照迁移拓扑的规定，将移出个体发送给目标岛屿的工作进程。最后，它会将这些个体发送给本岛屿上的所有工作进程，以便在全岛范围内停用已迁出的个体，然后再将其从自己的种群中停用。

在下一步中，工作进程会探查是否有来自其他岛屿的移入个体，并在适用时接收这些个体。随后，它会检查本岛其他工作进程迁出的个体，并尝试将其从自己的种群中停用。由于异步性，个体可能在进入种群之前就被指定为停用对象。Propulate 会在优化过程中持续修正这些同步伪影。

对于授粉（图 2），同一个体的副本可以同时存在于多个岛屿上。因此，工作进程可以从当前种群中的任意活跃个体中选择迁出的授粉者，且迁出时不会将其停用。为控制种群增长，授粉者会根据迁入策略替换目标种群中的活跃个体。为准确记录种群情况，目标岛屿的一名随机工作进程会选择要替换的个体，并通知其他工作进程。尚未进入种群的待停用个体会被缓存，留待下一次迭代时替换。此过程会持续进行，直到每个工作进程都评估完指定数量的代。最后，各工作进程之间的种群会同步，并返回最优个体。

Propulate 使用所谓的繁殖算子，从现有的亲代个体集合中繁育子代个体。它实现了多种标准遗传算子，包括均匀交叉、最优选择和最差选择、随机初始化、随机繁殖算子和条件繁殖算子、点突变和区间变异，以及多种形式的交叉。此外，Propulate 还提供了一种默认繁殖算子：从由固定数量当前适应度最高个体组成的繁殖池中随机选择两个父代个体，然后分别以指定概率执行均匀交叉和点突变

[1] 伪代码（包含迁移）和说明图见 github.com/Helmholtz-AI-Energy/propulate/tree/master/supplementary。

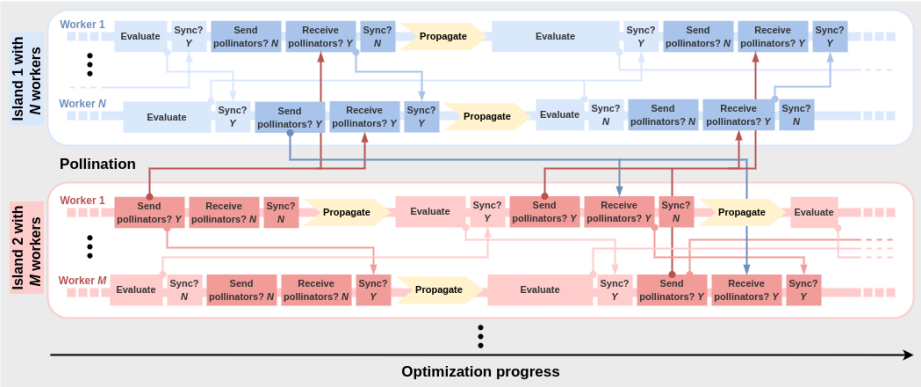

图 2. 异步授粉。考虑两个岛屿，分别有 N（蓝色）和 M（红色）个工作进程。我们通过追踪岛屿 1 上的工作进程 N 来说明授粉（深色）。完成评估和岛内更新（浅蓝色，见图 1）后，该工作进程执行授粉：将选定授粉者的副本发送给每个目标岛屿上的所有工作进程，此处为岛屿 2。目标岛屿上的工作进程异步接收授粉者（深蓝色箭头）。为正确记录种群情况，岛屿 2 上的工作进程 1 选出要替换的个体，并相应地通知该岛上的所有工作进程（中间的红色箭头）。随后，工作进程 N 接收来自岛屿 2、将纳入其种群的授粉者。接着，它会检查是否有个体在此期间已被岛上其他工作进程（此处为工作进程 1）替换并需要停用。完成这些与授粉相关的岛内种群更新后，它繁育出下一代。由于这一代没有发生授粉，它会直接接收来自岛屿 2 的授粉者。这次由工作进程 N 选择要替换的个体。

接着执行区间变异。为避免过早陷入局部最优，以指定概率添加一个随机初始化的个体，而不是从当前种群中繁育个体。

4 实验评估

我们在各种基准函数（见第 4.4 节）以及遥感分类中的一个超参数优化应用案例（见第4.5节）上评估 Propulate，后者提供了一个真实世界的应用场景。我们将结果与 Optuna 进行比较，因为它是使用最广泛的超参数优化软件。

4.1 实验环境

我们在卡尔斯鲁厄理工学院 Steinbuch 计算中心的分布式内存并行混合超级计算机 卡尔斯鲁厄高性能计算机 （HoreKa）上进行了实验。该系统的 769 个计算节点中的每个节点

表 1. 基准函数。

Name	函数	取值范围 全局最小值
球函数	$f_1 = x_1^2 + x_2^2$	$\pm 5.12 \quad f(0,0) = 0$
Rosenbrock 函数	$f_2 = 100\left(x_1^2 - x_2\right)^2 + (1 - x_1)^2$	$\pm 2.048 \; f(1,\ 1) = 0$
Step	$f_3 = \sum_{i=1}^{5}$ 积分 (x_i)	$\pm 5.12 \quad f(x_i \leq -5) = -25$
四次函数	$f_4 = \sum_{i=1}^{30}\left(i x_i^4 + \mathcal{N}_i(0,\ 1)\right)$	$\pm 1.28 \quad f(0, ..., 0) = \sum_i \mathcal{N}_i$
拉斯特里金函数	$f_5 = 200 + \sum_{i=1}^{20} x_i^2 - 10\cos(2\pi x_i)$	$\pm 5.12 \quad f(0, ..., 0) = 0$
格里旺克函数	$f_6 = 1 + \frac{1}{4000}\sum_{i=1}^{10} x_i^2 - \prod_{i=1}^{10}\cos\frac{x_i}{\sqrt{i}}$	$\pm 600 \quad f(0, ..., 0) = 0$
施韦费尔函数	$f_7 = 10V - \sum_{i=1}^{10} x_i$ 正弦 $\sqrt{\lvert x_i \rvert}$ 其中 $V = 418.982887$	$\pm 500 \quad f\left(x_1^*,\ ...,\ x_{10}^*\right) = 0,$ $x_i^* = 420.968746$
双球函数	$f_8 =$ 最小值 $\left(\sum_{i=1}^{30}(x_i - \mu_1)^2,\right.$ $\left.30 + s \cdot \sum_{i=1}^{30}(x_i - \mu_2)^2\right)$ 与 $\mu_1 = 2.5,\ \mu_2 = -\left(s^{-1}\left(\mu_1^2 - 1\right)\right)^{1/2},$ $s = 1 - \left(2\sqrt{50} - 8.2\right)^{-1/2}$	$\pm 5.12 \quad f(\mu_1, ..., \mu_1) = 0$
双 Rastrigin 函数	$f_9 = f_8 + 10\sum_{i=1}^{30} 1 -$ 余弦 $2\pi(x_i - \mu_1)$	$\pm 5.12 \quad f(\mu_1, ..., \mu_1) = 0$

均配备两颗 38 核 Intel Xeon Platinum 8368 处理器，基础频率为 2.4 GHz，最大睿频为 3.4 GHz；配备 256 GB（标准配置）或 512 GB（高内存和加速器配置）的本地内存、一块本地 960 GB NVMe SSD 磁盘以及两个网络适配器。其中 167 个节点为加速器节点，每个节点配备四块具有 40 GB 内存、通过 NVLink 连接的 NVIDIA A100-40 GPU。节点间通信采用低延迟、无阻塞的 NVIDIA Mellanox InfiniBand 4X HDR 互连网络，每个端口的速率为 200 Gbit/s。Lenovo Xclarity 控制器测量整个节点的能耗，但不包括文件系统、网络和冷却系统。操作系统为 RedHat Enterprise Linux 8.2。

4.2 基准函数

基准函数用于评估优化器的收敛性、准确性和鲁棒性。此类研究的参考价值，取决于我们对现实优化问题为何难以求解的特征的理解程度，以及将这些特征融入基准函数的能力 [28]。我们使用 Propulate 对多种传统及近期的基准函数进行优化，这些函数模拟了优化器在不同类型的问题中必须应对的情形（见表 1）。

– **球函数** 平滑、单峰、强凸且具有对称性，因此较为简单。– **Rosenbrock 函数** 在抛物线形谷底内部具有一个狭窄的极小值。– **阶梯函数** 表示平坦表面的问题。平台会给优化器带来障碍，因为它们缺乏有关哪个方向更有利的信息。– **四次函数** 是一种加入高斯噪声的单峰函数。由于它在同一点上永远不会返回相同的值，因此在此测试函数上表现不佳的算法，在噪声数据上也会表现糟糕。

表 2. 网格搜索参数。所有实验均使用 144 个 CPU，并平均分布在两个节点上。随机初始化概率指新个体完全通过随机方式生成的概率。

岛屿数量	2	4	8	16	32
岛屿种群规模	72	36	18	9	4
迁移（授粉）概率	0.1	0.3	0.5	0.7	0.9
授粉		True		假】【。	
交叉概率	0.1	0.325	0.55	0.775	
点突变概率	0.1	0.325	0.55	0.775	
随机初始化概率	0.1	0.325	0.55	0.775	

– **拉斯特里金函数** 是非线性的，并且具有高度多峰性。其表面由两个外部变量决定，这两个变量控制调制的振幅和频率。局部极小值位于大小为 1 的矩形网格上。它们的函数值会随着与全局最小值之间的距离增加而增大。

– **格里旺克函数**的乘积使子种群与并行遗传算法之间产生强依赖关系，而求和则形成一条抛物线。其局部最优值高于抛物线水平，但会随着维度增加而降低，也就是说，搜索范围越大，函数越平坦。

– **施韦费尔函数** 在远离全局最优解的位置有一个次优极小值。

– **Lunacek 双球函数**的 [28] 景观结构是两个二次函数的最小值，每个函数在搜索空间中形成一个单独的漏斗。两个球体沿搜索空间正对角线排列，其中最优球体和次优球体分别位于正象限和负象限的中间。它们之间的距离和屏障高度会随维数增加，从而形成全局不可分离的底层曲面。

– **Lunacek 双 Rastrigin 函数** [28]是拉斯特里金函数的双漏斗版本。该函数将全局结构单独作为影响一个广为人知的测试案例问题难度的主要差异。

4.3 元优化优化器

Propulate 本身也有影响其优化行为、准确性和鲁棒性的超参数。为了系统地探索这些超参数的影响，并给出透明的默认值建议，我们针对最重要的六个超参数进行了网格搜索。搜索空间见表 2。我们对四次函数、拉斯特里金函数和双 Rastrigin 函数进行了五轮网格搜索（参见表 1 和第 4.4 节），每轮使用不同的随机种子，且同一轮搜索中的所有网格点均使用相同的随机种子。这三个函数的全局最小值均为零。选择它们是因为它们的参数空间维度较高（分别为 30、20 和 30），且优化难度各不相同。对于四次函数，在五轮网格搜索涵盖的所有网格点中，Propulate 有 80.12% 找到低于 0.01 ± 0.005 的最小值。对于找到的最小值与全局最小值相差不超过 0.1 ± 0.05 的情况，这一比例升至 94.94%。相比之下，对于更复杂的函数，必须大幅放宽容差

对于更复杂的拉斯特里金函数和双 Rastrigin 函数，则是如此。在所有网格点中，拉斯特里金函数仅有 18.57% 的点的函数值低于 1.0 ± 0.5；而对于双 Rastrigin 函数，平均函数值低于 10 的网格点仅有一个。尽管双 Rastrigin 函数的平均函数值仅有一次低于 10，但我们发现，在每轮搜索中得到的最小值低于 1.0 的网格点占 3.31%。

在至少有一个结果小于 1.0 的网格点中，86.61% 使用了 16 个或 36 个岛，其余则使用了 8 个。由于 Propulate 会在搜索空间中的不同位置初始化不同的岛，岛的数量越多，其中某个岛处于非常有利位置的概率就越高。61.41% 的这些网格点采用了 0.7 或 0.9 的迁移概率，进一步证实了这一点。因此，如果某个岛的初始化位置较好，它很快就会通知其他岛。

所有最优网格点都采用授粉，这清楚地表明授粉优于实际迁移。为了确定其他超参数，我们计算了三个函数中排名前十的网格点结果的平均值。排名前十的网格点通过以下方法确定：按函数值的最低平均值和标准差分组，先按平均值排序，再按标准差排序。这样可以降低单次运行仅仅因初始随机种子有利而获益的可能性。交叉、点变异和随机初始化概率的平均值分别为 0.655 ± 0.056、0.363 ± 0.133 和 0.423 ± 0.135。岛的平均数量为 28.800 ± 6.009，对应的岛种群为 5.00 ± 1.043。平均迁移概率为0.527 ± 0.150。这些数值为选择 Propulate 的默认超参数提供了合理的起点（见表3）。由于网格搜索仅考察了参数相互独立的函数，我们认为较高的随机初始化概率可能有益，因为随机搜索具有优势 [6]。因此，我们将默认随机初始化概率降低至 0.2。由于这一现象也可能人为地降低迁移概率，我们将其默认值设为 0.7。交叉和点变异的默认概率分别设为 0.7 和 0.4。岛的规模设为 4 个个体。这是一个实用的选择，因为我们的测试系统每个节点有 4 个加速器，且每个节点的 CPU 数量是 4 的倍数。

4.4 基准函数优化

对于每个函数，我们使用相同的计算资源、并行度和评估次数，分别运行了十次等效的 Propulate 和 Optuna 优化。图 3 展示了优化精度随墙钟时间的变化，对比了采用网格搜索确定的默认参数（见表3）的 Propulate 与 Optuna 的默认优化器。就准确性而言，在大多数实验中，Propulate 和 Optuna 的表现相当。对于许多函数，例如施韦费尔函数、双 Rastrigin 函数和拉斯特里金函数，Propulate 甚至取得了更优的目标函数值。就墙钟时间而言，Propulate 始终快至少一个数量级。这是因为 Propulate 通过高速网络进行基于消息传递接口的通信，而 Optuna 使用基于 SQL 的关系数据库，且受限于速度较慢的文件系统。由于这些函数的评估成本较低，优化和通信占据了大部分墙钟时间。尤其是对于评估成本相较于

表3. 用于基准函数最小化的 Propulate 超参数。

岛屿数量	38
岛屿种群规模	4
授粉概率	0.7
交叉概率	0.7
点突变概率	0.4
西格玛因子	0.05
随机初始化概率	0.2
每个工作线程的迭代代数	256
选择策略	Best
授粉拓扑	全连接
迁移者数量	1
移出策略	Best
迁入策略	最差

搜索过程本身较低的问题，我们发现，Optuna 的计算效率会因其并行化策略中不可避免的频繁文件锁定而大幅下降，从而降低了其在大规模高性能计算应用中的可用性。

此外，我们还考察了 Propulate 和 Optuna 的种群随墙钟时间的演变情况。图 4 展示了最小化拉斯特里金函数的一个示例。Propulate 的速度大约快三个数量级，而且在目标函数值和与全局最优解的距离方面都取得了显著更大的进展。由于运行时间差异巨大，我们测得 Propulate 仅消耗 46.27 Wh，而 Optuna 消耗 2646.29 Wh。

4.5 遥感分类的超参数优化

BigEarthNet [35] 是一个用于遥感的 Sentinel-2 多光谱图像数据集。它包含 590326 个图像块，每个图像块都被赋予一个或多个 19 种 CORINE土地覆盖图标签 [10,35]。针对 BigEarthNet分类已训练了多个计算机视觉网络 [35], ，其中 ResNet-50 [20] 的准确率最高。此前曾使用一个较早版本的 Propulate 为此应用优化一组超参数和网络架构 [13],；当前版本采用了更通用、更高效的并行化策略，因此值得重新研究这一应用。与 [13], 类似，我们考虑不同的优化器、学习率（LR）调度器、激活函数、损失函数、每个卷积块中的滤波器数量以及激活顺序 [21]。搜索空间如表 4 所示。此外还包括优化器参数、学习率函数和学习率预热。我们只考虑基于SGD的优化器，因为它们具有共同的参数，因此不将类Adam优化器纳入搜索范围。我们推测，加入 Adam 导致了此前遇到的困难 [13]。如果验证损失连续十个训练轮次未能下降，就会提前终止训练。数据的准备方式与 [13]类似。该网络使用 TensorFlow 实现 [1]。

对于 Propulate 和 Optuna，我们都在 32 个 GPU 上各进行了三次搜索，每次持续 24 小时。我们使用 $1 - F_1^{\mathrm{val}}$，并将验证 F_1 分数作为需要最小化的目标函数。

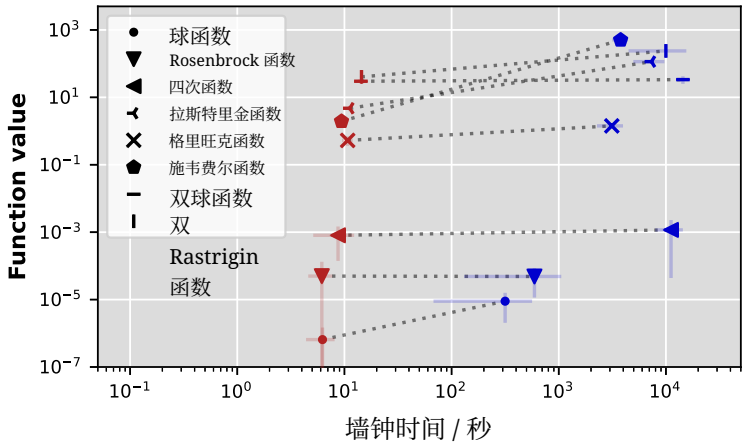

图3. 基准函数最小化的墙钟时间准确性。Propulate（红色）和 Optuna（蓝色）找到的最低函数值及达到这些值所需的墙钟时间，均为十次运行的平均值。未显示阶梯函数，因为两个优化器都能达到其最优值 -25 ，分别耗时 0.6 秒和 278.2 秒。

平均而言，Optuna 在 (7.05 ±3.14) 小时内取得了最优目标函数值 (0.39 ± 0.01)。Propulate 在 (5.30 ±2.41) 小时后便超过了 Optuna 的平均最优值，并在 (13.89 ± 5.15) 小时内取得了最优目标函数值 (0.36 ± 0.00)。

4.6 扩展性

最后，我们探讨了 Propulate 在第4.5节所述用例中的扩展性表现。图5展示了弱线性扩展和强线性扩展的结果。我们的基线配置使用了两个节点。由于每个节点配备四块 GPU，我们以八个工作进程为基准计算加速比和效率。在强扩展实验中，我们将评估总次数固定为 512，并增加工作进程数量，即 GPU 数量。我们对使用不同随机种子的三次运行取平均值，并在增加岛屿数量的同时保持每个岛屿有四个工作进程。加速比随着工作进程增至 128 而提升，此时达到最优值的大约一半。出现这种预期中的下降，是因为每个工作进程只处理少量个体，因此评估时间的方差会使较快的工作进程在最终种群同步前等待更久。此外，随着工作进程数量接近评估总次数，随机初始化的进化搜索也会逐渐趋近随机搜索。这意味着，搜索性能可能不如纯计算性能所暗示的那样理想。Propulate 仍可应用于如此大规模的计算，但必须相应调整其他搜索参数，如图5上方的弱扩展图所示。在规模最大的配置（128 个工作进程）下，弱扩展效率仅降至 95%。早期出现的超标量行为可能是由非顺序基线所致。

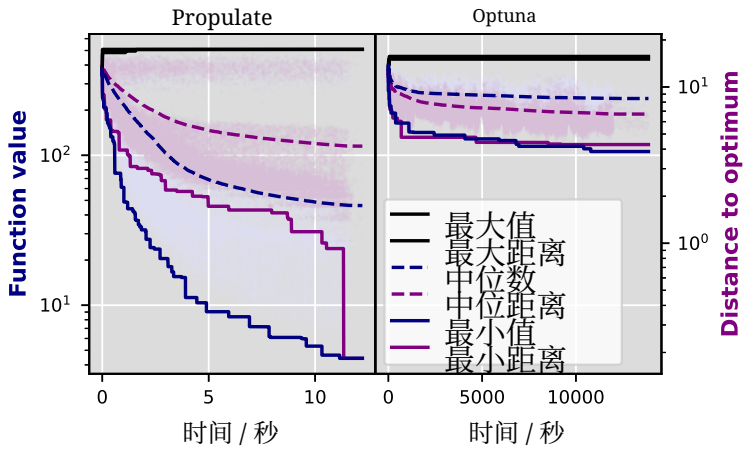

图4. 拉斯特里金函数的种群随墙钟时间的演变。Propulate（左）与 Optuna（右）的对比。目标函数值（蓝色）采用左侧刻度，到全局最优解的距离（紫色）采用右侧刻度。淡色圆点表示各个体的目标函数值/距离。实线（虚线）表示迄今达到的最小值（中位数）和距离。最大值（距离）以黑色显示。两种优化器均进行了 38 912 次评估。请注意时间轴上的差异。

5 结论

我们介绍了 Propulate，这是一种针对高性能计算进行适配的异步遗传优化算法及软件。实验评估表明，完全异步的评估、传播和迁移能够实现高效且可并行化的遗传优化。易用性虽比性能更难量化，却同样至关重要。尤其是在大规模高性能计算应用中，有些并行化和分布模型比其他模型更合适。像 Propulate 这样完全基于消息传递接口的实现，不仅对于高度并行且通信密集型的算法极其高效，而且易于部署和维护，因为高性能计算系统通常都提供所需的基础设施。我们研究的其他工具均不具备这一点，唯一的例外是未公开发布的 MENNDL。这也有助于在优化过程中更紧密地耦合个体，从而更高效地评估候选解；尤其是在神经架构搜索场景中，还能根据先前评估过的个体来进行提前停止。Propulate 已成功用于在不同高性能计算机上对各种机器学习模型进行超参数优化 [13,17]。未来工作的另一个方向是纳入可变长度的基因描述。这样就能高效地探索长度不同的互斥基因，例如我们神经架构搜索用例中 Adam 类和随机梯度下降类优化器的参数集。虽然目前已经可以实现这一点，但需要通过不便的变通方式，将非活动的

表 4. 用于 BigEarthNet分类 的 ResNet-50 超参数搜索空间。

优化器	优化器参数		学习率预热参数	
Adagrad	初始累积值	$[10^{-4}, 0.5]$	学习率预热步数	$[10^{0}, 10^{4}]$
SGD	梯度裁剪范数	$[-1, -1000]$	初始学习率	$[10^{-5}, 10^{-1}]$
Adadelta	梯度裁剪值	$[-1, 1000]$	衰减步数	$[10^{2}, 10^{5}]$
RMSprop	使用指数移动平均	布尔值	学习率预热幂	$[10^{-1}, 10^{1}]$
	指数移动平均动量	$[0.5, 1.0]$		
	指数移动平均覆盖	$[1, 10^{3}]$		
	动量	$[0.0, 1.0]$		
	Nesterov	布尔值		
	Rho	$[0.8, 0.99999]$		
	Epsilon	$[10^{-9}, 10^{-4}]$		

损失函数			学习率参数	
二元交叉熵	分类交叉熵	分类合页损失	衰减率	$[0.8, 0.9999]$
合页损失	KL散度	平方合页损失	阶梯反函数时间衰减	布尔值
激活函数			衰减率	$[0.1, 0.9]$
ELU	ReLU	Softplus	阶梯多项式衰减	布尔值
指数型 SELU		Softsign	末端学习率	$[10^{-4}, 10^{-2}]$
硬 Sigmoid Sigmoid		Swish	幂	$[0.5, 2.5]$
线性	Softmax	Tanh		

基因，并调整繁殖算子，以手动避免评估大量仅在非活动基因上有所不同的个体。

致谢

本研究由亥姆霍兹人工智能平台资助项目以及亥姆霍兹联合会在 HAICORE@KIT 分区上的倡议与网络基金支持。

参考文献

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., 等：TensorFlow：一种大规模机器学习系统。载于：第12届USENIX操作系统设计与实现研讨会（OSDI 16）。第265–283页（2016年）

2. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.：Optuna：新一代超参数优化框架。载于：第25届ACM SIGKDD知识发现与数据挖掘国际会议论文集。第2623 3330701 3 2631页（2019年）10 1145 3292500 –（）。https://doi.org/ . / . . Alba, E., Tomassini, M.：并行性与进化算法。《IEEE进化计算汇刊》**6**(5), 443–462（2002年）。https://doi.org/10.1109/TEVC.2002.800880

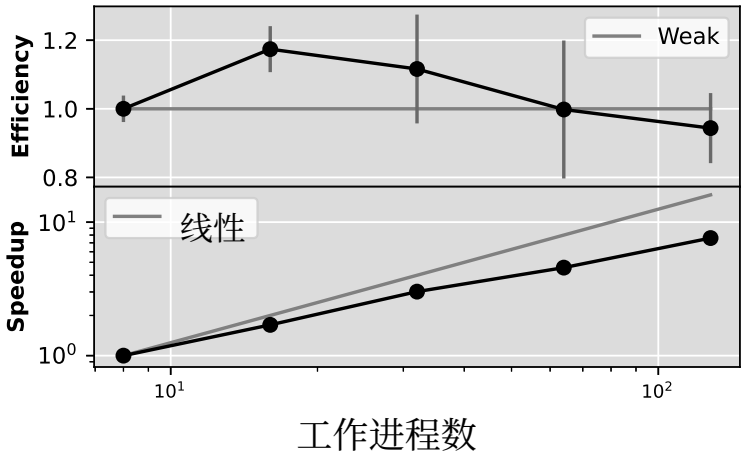

图5. 相对于八个工作进程基准的扩展性。弱扩展效率（上）和强线性加速（下）。用例和搜索空间见第4.5节。通过改变 OF 评估次数来调整弱扩展问题规模。结果为三次运行的平均值。

4. Alba, E., Troya, J.M.：并行分布式遗传算法综述。Complexity **4**(4), 31–52（1999年） 5. Authors, T.G.：GPyOpt：Python中的贝叶斯优化框架。http: // github.com/SheffieldML/GPyOpt（2016年） 6. Bergstra, J., Bengio, Y.：用于超参数优化的随机搜索。《机器学习研究杂志》**13**(10), 281–305（2012年）, http://jmlr.org/ papers/v13/bergstra12a.html 7. Bergstra, J., Yamins, D., Cox, D.：将模型搜索打造为一门科学：面向视觉架构的数百维超参数优化。载于：国际机器学习会议。第115–123页。PMLR（2013年）, http://proceedings.mlr.press/v28/bergstra13.pdf 8. Bianchi, L., Dorigo, M., Gambardella, L.M., Gutjahr, W.J.：随机组合优化元启发式算法综述。《自然计算》**8**(2), 239–287 4 9 2009 10 1007 11047 008 9098（）。https://doi.org/ . /s - - - . Blum, C., Roli, A.：组合优化中的元启发式算法：概述与概念比较。《ACM计算调查》（CSUR）**35**(3), 268–308（2003年）。https://doi.org/10 937505 10 1145 937503. / . . Bossard, M., Feranec, J., Otahel, J., 等：CORINE土地覆盖技术指南：2000年增补版，第40卷。欧洲环境署，哥本哈根（2000年） 11. Cant´u-Paz, E.：高效且精确的并行遗传算法，第1卷。施普林格科学与商业媒体（ 2000 10 1007 978 1 4615 4369 5 ）。12 https://doi.org/ . / - - - - . Cantu-Paz, E., 等：并行遗传算法综述。《并行计算机、网络与分布式系统》**10**(2), 141–171（1998年） 13. Coquelin, D., Sedona, R., Riedel, M., G¨ otz, M.：遥感分类问题中神经网络架构的进化优化。载于：2021年IEEE国际地球科学与遥感研讨会 IGARSS。第1587–1590页。IEEE（2021年）。https://doi.org/10.1109/IGARSS47720.2021.9554309

14. Elsken, T., Metzen, J.H., Hutter, F.: 神经架构搜索：综述。《机器学习研究期刊》 **20**(1), 1997–2017 (2019) 15. Feurer, M., Hutter, F.: 超参数优化。载于：自动机器学习，第3–33页。Springer, Cham (2019) 16. Fortin, F.A., De Rainville, F.M., Gardner, M.A.G., Parizeau, M., Gagn´e, C.: DEAP：让进化算法变得简单。《机器学习研究期刊》 **13**(1), 2171–2175 (2012) 17. Funk, Yannick and G¨otz, Markus, and Anzt, Hartwig: 利用深度学习预测稀疏线性系统的最优求解器。载于：2022年SIAM科学计算并行处理会议论文集。第14–24页。工业与应用数学学会 (2022)。https://doi.org/10 2 18 1137 1 9781611977141 . / . . . George, J., Gao, C., Liu, R., Liu, H.G., Tang, Y., Pydipaty, R., Saha, A.K.: 一种可扩展的云原生超参数调优系统 (2020)。https://doi.org/10 02085 19 48550 2006 . /arXiv. . . Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google Vizier：黑盒优化服务。载于：第23届ACM SIGKDD知识发现与数据挖掘国际会议论文集。第1487 3098043 20 1495 2017 10 1145 3097983 – 页 ( )。https://doi.org/ . / . . He, K., Zhang, X., Ren, S., Sun, J.: 用于图像识别的深度残差学习。载于：IEEE计算机视觉与模式识别会议论文集。第770–778页 (2016) 21. He, K., Zhang, X., Ren, S., Sun, J.: 深度残差网络中的恒等映射。载于：欧洲计算机视觉会议。第630–645页。Springer (2016)。https://doi.org/ 10 1007 978 3 319 46493 0˙38 . / - - - - . Hertel, L., Collado, J., 22 Sadowski, P., Baldi, P.: Sherpa：面向机器学习模型的超参数优化。第32届神经信息处理系统会议 (NIPS 2018) (2018)，https://github.com/sherpa-ai/sherpa 23. Holland, J.H.:《自然与人工系统中的适应：关于生物学、控制与人工智能应用的入门分析》。MIT Press (1992 0001 24 10 7551 1090 001)。https://doi.org/ . /MITPRESS/ . . . Hutter, F., Hoos, H.H., Leyton-Brown, K.: 面向通用算法配置的序贯模型优化。载于：学习与智能优化国际会议。第507–523页。Springer (2011)。https://doi.org/ 10 1007 978 3 642 25566 3˙40 . / - - - - . Koch, P., Golovidov, O., 25 Gardner, S., Wujek, B., Griffin, J., Xu, Y.: Autotune：用于超参数调优的无导数优化框架。载于：第24届ACM SIGKDD知识发现与数据挖掘国际会议论文集。第443 3219837 26 452 2018 10 1145 3219819 – 页 ( )。https://doi.org/ . / . . Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I.: Tune：用于分布式模型选择与训练的研究平台。arXiv预印本 arXiv:1807 05118 27 05118 2018 10 48550 1807 . ( )。https://doi.org/ . /arXiv. . . Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: SMAC3：用于超参数优化的多功能贝叶斯优化软件包。《机器学习研究杂志》**23**, 54–1 (2022) 28. Lunacek, M., Whitley, D., Sutton, A.: 全局结构对搜索的影响。载于：自然启发的并行问题求解 – PPSN X, LNCS 5199。第498–507页。Springer (2008 10 1007 978 3 540 87700 4˙50)。 29 https://doi.org/ . / - - - - . Luque, G., Alba, E.: 并行遗传算法：理论与实际应用，第367卷。Springer (2011)。https://doi.org/10.1007/978-3-642-22084-5

30. Mitchell, M.: 遗传算法导论。MIT Press (1998) 31. Rapin, J., Teytaud, O.: Nevergrad——无梯度优化平台。
https://GitHub.com/FacebookResearch/Nevergrad (2018) 32. Snoek, J., Larochelle, H., Adams, R.P.: 机器学习算法的实用贝叶斯优化。载于：Pereira, F., Burges, C., Bottou, L., Wein- berger, K. (编) 神经信息处理系统进展。第25卷。Curran Associates, Inc. (2012)，https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf 33. Song, X., Perel, S., Lee, C., Kochanski, G., Golovin, D.: Open Source Vizier：用于可靠、灵活黑盒优化的分布式基础设施与应用程序编程接口。载于：自动化机器学习会议，系统专题 (AutoML-Conf 系统) (2022)，https://github.com/google/vizier 34. Sudholt, D.: 并行进化算法——《计算智能手册》章节。Springer (2015)。https://doi.org/10.1007/978-3-662-43505- 2˙46
35. Sumbul, G., Kang, J., Kreuziger, T., Marcelino, F., Costa, H., Benevides, P., Caetano, M., Demir, B.: 用于遥感图像理解、采用新类别命名体系的BigEarthNet数据集。arXiv预印本 arXiv:2001.06372 (2020)。https://doi.org/10 06372 36 48550 2001 . /arXiv. . . Toklu, N.E., Atkinson, T., Micka, V., Srivastava, R.K.: EvoTorch：直接构建于PyTorch之上的高级进化计算库，由NNAISENSE创建。
https://github.com/nnaisense/evotorch (2022) 37. Tomassini, M.: 空间结构化进化算法：时空中的人工进化。Springer (2006 6 38 10 1007 3 540 29938)。https://doi.org/ . / - - - . Wang, J., Clark, S.C., Liu, E., Frazier, P.I.: 昂贵函数的并行贝叶斯全局优化。《运筹学》**68**(6), 1850–1865 (2020)。https://doi.org/
10 1287 2019 1966 . /opre. . . Weiel, M., Gotz, M., Klein, A., 39
Coquelin, D., Floca, R., Schug, A.: 使用灵活目标函数对生物分子模拟参数进行动态粒子群优化。Nature Machine Intelligence **3**(8), 727–734 (2021)。https://doi.org/10 3 40 1038 42256 021 00366 . /s - - - . Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.H., Patton, R.M.: 通过进化算法优化深度学习超参数。载于：高性能计算环境中的机器学习研讨会论文集。第1–5页 (2015)。https://doi.org/10.1145/2834892.2834896