arXiv:2304.03235v1 [cs.NE] 6 Apr 2023

缓存未命中更少的 GI 软件*

W.B. Langdon、Justyna Petke、Aymeric Blot、David Clark {W.Langdon,j.petke,david.clark}@ucl.ac.uk aymeric.blot@univ-littoral.fr 伦敦大学学院计算机科学系

摘要

缓存虽因其名称而常被忽视，却在现代乃至未来硬件的性能中发挥着至关重要的作用。我们使用 MAGPIE（通过软件进化实现机器自动化通用性能改进）证明，遗传改进（GI）能够减少现有计算机程序的缓存负载。Magpie 以 C 和 C++ 源代码行为操作对象，通过局部搜索生成新的功能等效变体，从而减少 L1 数据缓存未命中。我们在两个工业级开源程序（Google 的开放位置代码 OLC 和 Uber 的六边形分层空间索引 H3）以及两项二维照片图像处理任务——像素计数和 OpenCV 的 SEEDS 分割算法——上测试了缓存未命中的减少效果。

Magpie 的补丁在功能上实现了泛化。在一个案例中，它们将性能最高的 L1 缓存上的数据未命中率大幅降低了 47%。

CCS 概念

• **软件及其工程 → 基于搜索的软件工程**。

关键词

遗传编程, 遗传改进,SBSE, 线性表示, 软件弹性, 自动代码优化, 禁忌 , 非平稳噪声 , perf, 全球位置 , Plus Codes, 邮政编码 , OpenCV, 图像分割

1 引言

Jack Dongarra 于 2021 年获得图灵奖。为此，在最近发表于《美国计算机学会通讯》的一篇文章中 [10] ，他描述了高性能计算当前及可预见的未来所面临的限制：过去二十年来，时钟频率并未显著提升，但摩尔定律 [33] 仍持续推动晶体管数量呈指数级增长。Dongarra 认为，这将继续推动当前趋势，使硬件并行度不断提高。他还指出，计算硬件的瓶颈如今已不在处理器速度，而在于将数据传送到计算引擎所需的时间。也就是说，计算成本低，数据移动才昂贵。

尽管这一趋势十分普遍，但硬件并行性日益增强的趋势，可通过显卡（GPU）及类似的加速器架构（例如 TPUs）得到 exemplified；如今，这些架构已成为超级计算机和深度神经网络训练的骨干。长期以来，nVidia 一直不愿引入通用数据缓存而不是将其交给显卡，始终坚持认为电脑游戏的开发者会了解其程序内部的数据流，因此能够优化软件，以充分利用 GPU 所提供的巨大数据带宽。这种想法一直都不切实际，即使对于经验非常丰富的程序员来说，要获得最佳性能也非常困难。因此，如今为了让性能编程变得更加容易，GPU 制造商夸耀说，他们的 GPU 包含多级缓存。

Dongarra试图反对使用缓存来让软件开发者更轻松的趋势，并说道：“不能仅仅依赖硬件缓存，必须设计新的算法” [10,页 68]。然而，他没有说明这将如何实现。遗传编程在发明新解决方案方面有着悠久而成熟的传统 [17]。与GP不同，遗传改进（GI）并非每次都从头开始进化，而是 [23, 38] 在现有软件上寻找更新。尽管遗传改进主要用于修复计算机软件缺陷 [12] 或加速代码 [24], ，我们希望通过展示这样一个事实来强化这一主张：尽管存在噪声，GI 工具Magpie 有时能够提高传统 Intelx86 台式计算机中甚至最快级别数据缓存的有效性，即使面对经过编译器优化的代码[1]也是如此。因此，GI和 GP 或许可以针对如何在高度并行的硬件中有效利用未来的数据通路这一问题展开双管齐下的攻势：GP负责发明新算法，而 GI 则调整现有代码，以更好地利用新型硬件。

第 3 节详细介绍了 Google 的 OLC 和 Uber 的 H3 数字制图程序、Blue 图像基准，以及 OpenCV SEEDS 资源密集型图像分割算法的适应度测试用例。第 4 节则介绍了如何运用优惠券收集问题的论证，来选择 Magpie 应采样多少搜索空间。第 5 节给出结果（另见表 1）；接着，第 6 节简要介绍 Magpie 对 C 和 C++ 源代码所做的更改。在图像示例中，即使面对使用现有编译器（GCC -O3）优化过的代码，这些更改也能减少 L1 数据缓存未命中；本节还讨论了非平稳噪声。第 7 节总结指出，Magpie 已可投入使用，并且在此将 L1 数据缓存未命中最多减少了 47%。在所有四个示例中，补丁都能在功能上泛化，但只有在数据密集型图像处理示例中，我们才观察到持续的缓存减少（见图 1、4、7、8、9 和 10）。不过，我们先介绍背景。

*W.B. Langdon、Justyna Petke、AymericBlot、David Clark 的长篇版本。2023 年。通过进化改进的软件：更少的数据缓存未命中。载于遗传与进化计算大会伴会（GECCO '23 Companion），2023 年 7 月 15 日至 19 日，里斯本，ACM。https://doi.org/10.1145/3583133.3590542

[1]CPU 硬件同时提供数据缓存和指令缓存。两者对于性能都至关重要。然而，由于局部性通常较好，而且程序机器码相较于其处理的数据规模较小，指令缓存现在乃至未来可能都不如数据缓存那么成问题。在这些实验中，L1 指令缓存为 32KBytes，指令缓存未命中仅约百次；而例如使用 SEEDS 时，即使是图 3 这样的小图像，L1 数据缓存未命中也超过 40000 次。

2 背景

计算机具有普适性。一切都依赖于计算机。即使在今天，许多最富有的人之所以富有，也是因为他们在世纪之交左右创办了非常成功的软件公司。总体而言，信息技术，尤其是软件，渗透到了第三个千年，并将继续占据主导地位。事实上，整个星球都对软件上了瘾。尽管编程已有60 多年的历史，软件仍然由人工编写。在很大程度上，自动编程仍然只是一个梦想。

在 GECCO-2009，StephanieForrest、ThanhVu Nguyen、WesWeimer和 Claire Le Goues [12] 展示了遗传编程 [5, 16, 39, 40, 42, 43] 能够自动修复计算机软- 件中的软件缺陷 [28, 44]。人工智能（AI）首次被应用于软件工程中的一个重大问题，即工业规模的程序 [1]。此后，自动程序修复（APR）领域蓬勃发展 [32]。受Stephanie等人的启发，我们 [23, 24]开始将遗传编程应用于以多种方式改进人工编写的软件，而不仅仅是修复软件缺陷 [38]。

尽管遗传编程仍然是遗传改进（GI）中常见的搜索技术，但局部搜索正日益受到欢迎 [6]。除了功能日益强大的计算机之外，GI 能够实现扩展的原因还在于，它不会在每次运行时都从头开始，而是建立在现有软件的基础之上。另一个巨大的优势是，GI可以利用现有的、经过艰苦手工编写的代码进行自动双重检查。如同回归测试 [14], 一样，现有程序实际上成为了自身的规范。也就是说，它可以作为测试预言机，用于自动进行功能和非功能改进。这可以是运行时性能（无论是经过的时间、内存需求等），也可以是其功能。我们上文提到过自动修复错误的成功，但功能改进还可以包括让程序给出更准确的答案 [26]。进化还被用于使网络软件适应色盲用户，以及帮助为聋人患者调试助听器 [29]。自动生成的软件测试 [13] 日益取得成功，也自然推动了遗传改进的发展。

遗传改进研究通常通过定制的一次性实验进行。David White 认识到了这一点，并提出了 GIN [46] 作为用于Java程序的通用遗传改进工具。Gabin An [3, 4] 提出了适用于Python的PyGGI。两种工具都得到了广泛使用和更新： [8, 30, 36, 37] 和 [2, 6, 15, 31, 41], ，此外还提出了更多遗传改进工具 [18]。然而，最近一项用户研究表示，遗传改进缺乏用户友好的工具 [47]。

2.1 Magpie

针对这一点，Aymeric Blot 编写了Magpie，它不仅易于用户使用，还结合了GI 和参数调优。它于去年作为一个开源项目发布。与 PyGGI 2.0 [2] （Magpie 正是基于它开发的）一样，它可以从 GitHub[2] 免费获取。我们使用它，希望能够说服读者：进化原则上可以改进软件质量的任何<style id='3'>可比度量。虽然它同样使用 Python 编写，但其目标是支持任何计算机编程语言。它主要在 AppleMac和 Linux 笔记本电脑上经过测试，但其目标是足够通用，可以在 MicrosoftWindows下运行。在这里，我们在 Linux桌面电脑上对其进行测试，并未尝试维护与 Microsoft 的兼容性。

[2]https://github.com/bloa/magpie

截至 2022 年 11 月 27 日，Magpie 包含 4871 行代码（包括示例和文档），大部分使用 Python 编写。它包含以 Python、C、C++ 和 Ruby 编写的示例。

2.2 对 Magpie 的更新

在之前的工作中 [22], 我们增强了 Magpie，使其能够使用 Python 的 ctypes直接调用补丁代码。这样，我们可以直接在单个C/C++ 例程上收集数据，而不是针对整个Magpie子进程，从而减少噪声。此外，为了将 Python 解释器排除在测量之外，我们在调用补丁代码之前，通过设置并读取一个固定的 32K 字节数组来清除 L1数据缓存（即我们的CPU的L1 数据缓存大小）。为了使适应度更加稳健，我们会多次测试每个补丁；由于均值对于嘈杂的离群值来说一向并不可靠，因此我们改用第一四分位数来概括测量结果。

在图像分割示例（第 3.2节）中，17% 的行由单个右大括号 } 组成。这些行自然是可以互换的，因此人们发现Magpie 浪费了大量精力来测试实际上完全相同的程序（除了空白等差异）。为避免这种情况，编译步骤会维护一个“禁用”列表 [25]。此前 [25] 我们维护的是一个包含基因型和表现型信息的复杂禁用列表（大小最多可达340 MBytes），而在这里，我们只需保留每个目标文件的副本（每次运行平均有407 个文件，占用约30 MBytes）。编译完成后，通过将补丁的目标文件与先前补丁中大小相同的目标文件进行简单比较，即可拒绝语义上相同的补丁。新的唯一补丁会被添加到禁用目录中，而相同的补丁则会被丢弃，不进行适应度测试。尽管从理论上讲，随着禁用目录不断增大，这一过程可能会变慢，但实际上所耗费的时间微不足道。

Linux GNU perf 实用程序允许访问许多硬件性能计数器。具体来说，我们使用 perf 运行时库 linux/perf_event.h 收集 L1 数据缓存和指令缓存未命中的次数、已执行指令的计数以及经过的时间。（适应度函数仅使用 L1 缓存数据未命中次数。）

2.3 数据集背景

我们使用了四个开源 C/C++ 示例。其中两个是工业地理空间程序，均以 C [21]编写：一个来自 Google 的 OLC，另一个来自 Uber 的 H3 （参见第 3.1 节和图 1 ）。另外两个是 C++ 摄影图像处理示例（第 3.2 节），分别是我们在 2018 年 Tarot 暑期学校使用的 Blue 基准测试 [19]（图 2），以及 OpenCV 的图像分割代码 [27] （图 3 和图 4）。

3 适应度函数

Magpie 尝试在所有测试用例上运行打过补丁的程序。如果补丁通过了所有测试，Magpie 会再次多次运行该程序，以尽可能准确、稳健地估算其性能。

总之，Magpie 使用多个目标来计算变异的适应度，按顺序（优先级）依次为：1）补丁能否无错误地编译（忽略警告）；2）变异后的软件能否在每个测试用例上运行，且不发生崩溃或超时；3）其输出是否与原始代码的输出相同；4）perf 记录了多少 L1 数据缓存未命中。

(1) 源代码使用GCC 编译器（版-本10.2.1）进行编译，采用相同的选项和开关，例如 -O3，正如

开发者，避免浪费时间报告多个错误，使用 -fmax-errors=1 用于让GCC在第一个错误处停止。如果编译器成功编译补丁，则将其与补丁外不可演化的代码，以及调用 perf 运行时库的 C 代码链接起来，形成一个共享对象库。（使用 GCC 选项 -shared 和 -fPIC 创建共享库 prog.so。）Python解释器使用Python ctypes 调用共享库中的 C接口例程，该例程调用 perf 运行时库和经过补丁修改的代码。补丁运行后，perf运行时库从CPU内部提取硬件计数器，接口代码将这些计数器连同补丁生成的输出一起传回 Python 解释器。

(2) Magpie（通过 Python解释器）和变异体本身都可以通过 Unix退出状态来发出问题信号。在任一情况下，Magpie 主线程都会丢弃适应度较差的补丁，然后继续生成并测试下一个补丁。

(3) 对于每个测试用例，Python 子进程都会检查补丁的输出是否符合每个用户提供的测试用例的预期。例如，对于OLC，Magpie 会检查补丁后的代码是否针对测试用例中的纬度和经度对，返回了与Google 的代码相同的16个字符。如果有任何字符不同或缺失，测试就会失败，并且该补丁的适应度测试会立即停止。在使用Magpie 之前，我们在每个测试用例上运行了原始的OLC、H3、Blue 和 SEEDS 程序，记录它们的输出，然后将其自动转换为Python列表数据结构。例如，对于 OLC，适应度训练由十个纬度和经度浮点数以及十个 16 字符字符串组成，其格式为十个 Python 方括号元组。

(4) 上文第 2.2 节描述了如何将 perf C 运行时库集成到 Python 中。Magpie 使用补丁所有重复测量值的第一四分位数（Q1）来给出其适应度度量（另见第 2.2 节）。即使在被认为具有确定性的程序中，用于缓存统计信息、已执行指令数和经过时间的硬件计数器也存在噪声。尽管对 perf 的 L1 数据缓存未命中采用了 Q1 等稳健统计量，适应度仍然存在噪声。但正如我们将看到的，在某些情况下，Magpie 能够取得进展。

3.1 Google 的 OLC 和 Uber 的 H3 测试用例：英国邮政编码

我们使用了先之前相同的测试用例 [21]。

Google 的开放位置代码（OLC）https://github.com/google/open-location-code（下载于 2022 年 8 月 4 日）和 Uber 的六边形分层地理空间索引系统（H3）https: //github.com/uber/h3（下载于前一天）都是开放的行业标准（源代码总规模分别为 OLC 14 024 行、H3 15015 行）。它们都包含可将纬度和经度转换为各自内部代码的 C 程序（见表 1）。对于 OLC，我们使用 Google 的 16 字符编码；对于 H3，我们使用 Uber 的最高分辨率（-r 15），该分辨率使用 15 个字符。继之前的工作之后 [21],，我们使用实际地址的位置作为测试用例。

对于 Google 的 OLC，我们使用了 [21] 数据集，该数据集包含从 https:// www.getthedata.com/downloads/open_postcode_geo.csv.zip 下载的前一万个英国邮政编码的位置（日期为

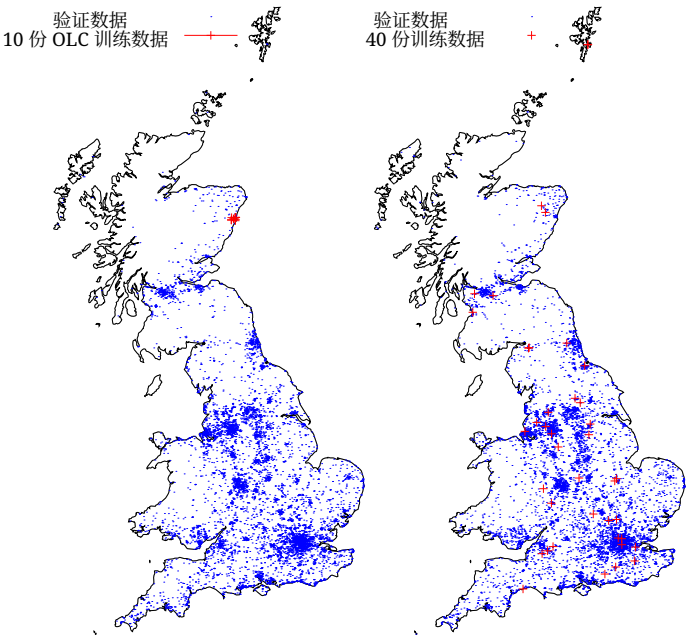

图 1：左：在 Aberdeen 附近随机选取的十个 OLC 训练点（红色）。留出集（蓝色点）为英国邮政编码。右：从十个 H3 运行时类别中随机选取的四十个训练点。留出集（蓝色点）为一万个随机英国邮政编码的位置（与 H3 训练数据或 OLC（左图）留出数据不重叠）。OLC 和 H3 补丁通过了全部 10000 项留出测试。数据集来自[21]。

2022年3月16日）。这些地址按字母顺序排列，从AB10AA开始，该地址位于阿伯丁。训练时，均匀随机地选取了十对纬度和经度（见图1）。未变异的代码在每一对数据上运行，并保存其输-出（16字节）。对于每个测试用例，将每个变异体的输出与原始输出进行比较。

Uber 的 H3 也采用了类似处理（见图 1 右侧）

3.2 Blue 和 OpenCV SEEDS 的测试用例

此前，我们为参加 2018 年 TAROT 软件测试、验证与确认暑期学校的学生制作了一个 GISMO GI 系统的简单示例 [19] http:// www.cs. ucl.ac.uk/staff/W.Langdon/ftp/gp-code/opencv_gp.tar.gz。我们生成了十张随机图像（见图 2），并计算了每张图像中“蓝色”像素的数量。Magpie 将这些图像用作训练数据，目标是优化代码，以尽量减少 L1 缓存未命中。请注意，训练图像包含 $96 \times 128 = 12\,288$ 个彩色像素，占用 49 152 字节，因此超出了 L1 数据缓存。我们删除了注释，留下 100 行 C++ 代码。

相比之下，OpenCV 是一套庞大的 C++ 图像处理工具。在 2023 年初，OpenCV 在 GitHub 上的开源代码库包含超过两百万行代码（主要使用 C、C++ 和 XML 编写）。因此，我们选取了一个重要的例程：先进的 OpenCV SEEDS 超像素图像分割算法。该算法曾参与 5 万美元的 OpenCV 挑战赛，



图 2：十个“Blue”随机 96×128图像之一。该示例包含 2135 个蓝色像素 [19]。



图 3：OpenCV SEEDS 图像分割训练204×153图像。下载自 http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/opencv_gp.tar.gz [27]。

我们此前也曾在 GI 实验中使用它，以在遵循其应用程序接口 [27] 的同时缩短运行时间。

SEEDS 超像素算法的 OpenCV 代码包含 1500 行 C++ 代码，但重要的例程都集中在一个文件 updatePixels.cpp 中。删除注释和空行后，共有 319 行。不幸的是，SEEDS 算法的计算量很大，因此我们没有使用从 [27], 获取的完整图像，而是将训练数据缩减至 1/16（见图 3）。请注意， 204×153训练图像包含 31212 个彩色像素（124848 字节），因此 SEEDS 算法使用的主要数据结构超出了 L1 数据缓存的容量。

4 MAGPIE 搜索

Magpie 使每个实验的用户自定义参数与通用代码清晰分离。

程序运行时常会经历一段“稳定期”，在此期间运行时间比平常更长。Magpie 通过先进行“预热”来解决这个问题。在开始任何搜索之前，Magpie 会生成空补丁并对其进行测试。（Magpie 最初生成 4 个空补丁，现在使用 11 个。）预热补丁的适应度会被丢弃，但 Magpie 会将之后所有补丁的性能报告为其相对于平均预热适应度的比例。因此请注意，Magpie 的适应度值越低越好。

除 11 个预热补丁外，Magpie 还生成了 700 个 OLC、19 077 个 H3 、 904 个 Blue 和 3151 个 SEEDS 补丁（见表 1 ）。



图 4：示例留出 2448×3264 图像（在 100% 12、14、18和 116尺寸下测试）。这是从 http://www.cs.ucl.ac.uk/staff/W.Langdon/egp2014/granada/ 随机选取的 30 个 SEEDS 示例之一。自动演化得到的 SEEDS C++ 代码在五种分辨率下分别进行了多次验证。

表 1：左：待优化的 C/C++ 源代码大小（已移除注释和空行）。中：最多五次 Magpie运行的平均值。第3–4 列为补丁大小。第5 列为最佳适应度（L1 数据缓存未命中的平均减少量）。右：第6 列为已探索的搜索空间大小。第 7列为能够编译、正常运行并给出正确答案的变异体比例。第 8列为在配备GCC 10.2.1 和Python 3.10.1的 3.6 GHz Intel i7-4790 台式机上使用 1 个核心运行 Magpie 的平均耗时。

示例	LOC	突变体 size	突变体 L1D	Magpie 步骤	Magpie % 运行正常	Magpie 持续时间
OLC	134	1– 5	5%	700	23%	66 秒
H3	1615	6–18	6%	19077	33%	3.9 小时
blue	100	7–10	47%	904	30%	1.7 小时
SEEDS	319	2– 7	7%	3151	2%	5.2 小时

OLC和H3（700和19077）值取自我们之前的工作 [21]。和之前一样 [21], , 我们使用优惠券收集者 [11] 论证来计算需要多少随机样本，才能几乎确定至少访问 C/C++ 源代码的每一行一次。（要优化的 H3 源代码比其他代码大得多，见表 1，因此需要更大的搜索 ef-fort。）在这四种情况下，我们使用了 Magpie的运行时缩减选项：python3 -m bin.magpie_runtime。

Magpie 在一台配置为 32 GB、3.60 GHz Intel i7-4790 的台式机上使用单个主线程运行；该机器运行联网的 Unix Centos 7，其他方面大多处于空闲状态，使用 Python 3 版本 3.10.1 和 GNU C 编译器 10.2.1 版。在所有四种情况下，perf 记录的数值都有很大波动。（例如，参见图 7 和图 8。）

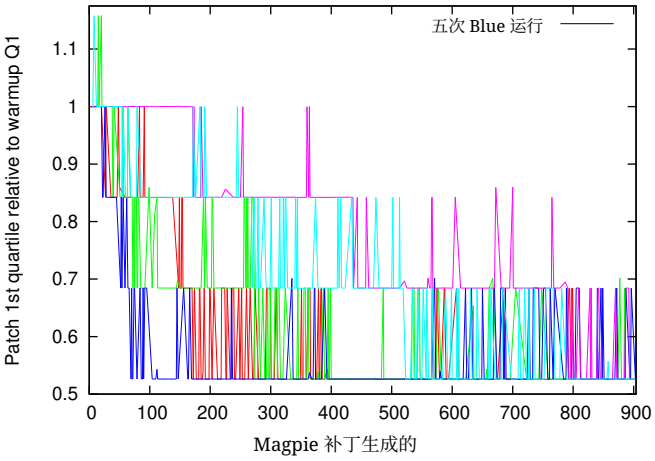

图 5：Blue 补丁。五次 Magpie 运行相对于预热的 perf L1D 计数。图中仅绘制通过全部十个训练用例的补丁的适应度（越低越好）。

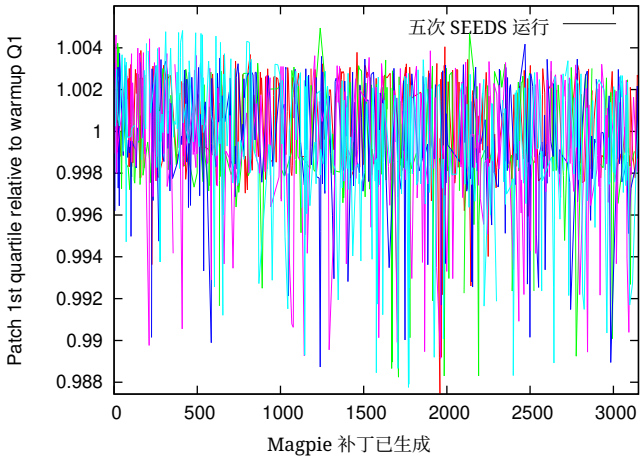

图 6：SEEDS 补丁。五次 Magpie 运行相对于预热的 perf L1D 计数。为避免图表杂乱，不绘制适应度较差的数据，即高于 1.005 的数据。（适应度越低（纵轴）越好。）

5 个结果

结果汇总于表 1，图 5 和图 6 展示了 Magpie 运行期间找到的 Blue 和 SEEDS 补丁的适应度。请特别注意，第 3 列和第 4 列（变异体大小和 L1D）中的结果，指的是 Magpie 在运行期间找到的最佳适应度。也就是说，表 1 中的 L1D 改进是训练期间测得的。对于两个地理类程序（OLC 和 H3），尽管补丁仍能通过多达 10000 项留出测试，但预期的缓存性能提升并未泛化。事实上，消除噪声后，原始代码与补丁代码的 L1 数据缓存未命中似乎并无实质差异。这与另外两个数据丰富的图像程序形成鲜明对比：对于大小与训练数据相同的图像，补丁确实减少了数据缓存未命中（见图 9 和图 10）。同样，Blue 和 SEEDS 的补丁在泛化方面也表现良好：面对未见过的图像，仍能给出正确答案。然而，图 10 右侧（蓝色 ×）显示，对于尺寸超过训练图像四倍的图像（图 3），SEEDS 补丁并未减少 L1 数据缓存未命中。

平均而言，72% 的 OLC 补丁无法编译3，约 5% 的补丁无法通过一个或多个测试用例，而剩余的 23% 通过了全部十项适应度测试。表 2 总结了全部四个实验的统计数据。H3 和 Blue 的模式相似。然而，SEEDS 使用的禁忌列表意味着，每当补丁未通过禁忌检查时，Magpie 都会将其标记为无法编译。因此，SEEDS 的 ok 列数值较低[4]。请注意，大多数补丁都能编译、正常运行并通过所有测试。

尽管 Magpie 有一个很实用的工具，可以缩减补丁，但由于基于 perf 的适应度度量存在较多噪声，我们没有使用该工具。

在全部四个实验中，最终生成的补丁与原始程序产生了相同的结果。对于其中两个地理工具

[3]此前，我们使用 LLVM IR 的专用变异算子，确保所有变异体都能成功编译为机器码 [21]。[4]在 [20] 中，我们采用了类似的思路，通过检查 GNU gcc 编译器生成的 X86 汇编代码，测试变异后的代码是否相同。Mike Papadakis 等人也在 [35] 中比较了编译器输出，以寻找等价变异体。

表 2：Magpie 补丁汇总。每个 L1 数据缓存实验五次运行的平均值。

	编译错误	测试失败	超时	过大	ok
olc	72%	5%	0%	0%	23%
h3	61%	4%	0%	2%	33%
blue	56%	13%	1%	0%	30%
seeds	87%	10%	0%	0%	2%

（OLC 和 H3），留出集包含“缺失数据”，即没有纬度、经度位置的邮政地址。在这 85 个案例中，OLC 生成默认输出，而 H3 则中止运行（返回非零 Unix 错误代码）并显示错误消息。H3 的补丁同样检测到了该错误并进行了报告。在其余 9915 个留出位置上，补丁同样返回了与原始 H3 相同的输出。也就是说，OLC 和 H3 的补丁都通过了全部 10 000 项留出测试。然而，在这两种情况下，我们都无法证明 Magpie 运行中观察到的适应度能够通过重新运行这些实验得到。（另见第 6.1.1 节和第 6.1.2 节。）

我们的结果汇总于表 1 第 5 列“L1D”，该列给出了训练期间 L1 数据缓存未命中率的降低百分比。不幸的是，正如下一节将讨论的那样，使用 OLC 和 H3 训练时报告的缓存使用率改进并未泛化；在样本外测试中，L1 数据缓存未命中并没有减少。相比之下，在两个图像处理示例中，我们发现 L1 数据缓存未命中显著减少（非参数曼-惠特尼检验）。对于 Blue 基准测试，降幅为 47%。而对于 OpenCV 图像分割算法 SEEDS 的补丁，在 30 张与训练图像大小相同、未见过且无关的图像上，降幅为 1%。C++ 代码

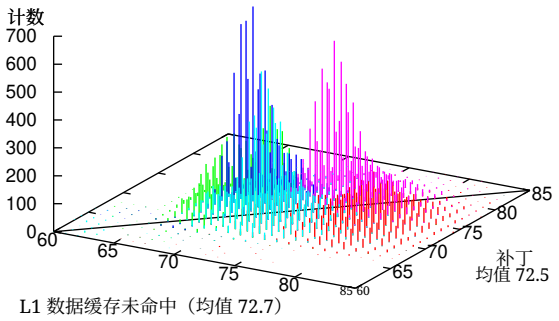

图 7：Magpie OLC 补丁在 10 000 个留出邮政地址上的样本性能。补丁通过了全部 10 000 个样例。与 H3 不同，OLC 能够顺利处理这 85 个无效位置而不会出错。测试在同一台计算机上重复进行了五次。五种颜色表示五个不同的分布。每次重复中，原始版本与补丁的分布都有显著差异，但这是一个伪象，见第 6.2 节。图 1 左侧展示了 10 个训练数据位置和这 10 000 个留出邮政编码。

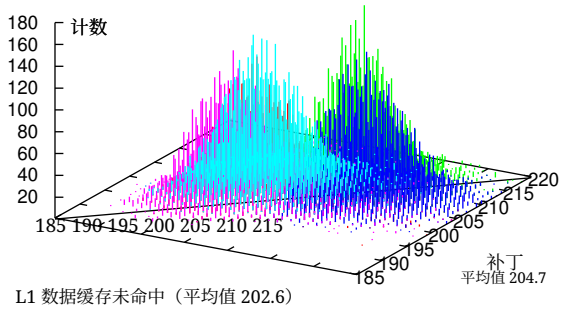

图8：Magpie H3 补丁在 9915个留出邮政地址上的样本外性能。该补丁通过了全部10000 个示例。其中有85 个无效位置（对于这些位置，补丁会正确生成错误消息），因此在本比较中予以排除。该补丁平均额外产生2.1 次 L1 数据缓存未命中。请注意数据 σ=5的分布。与图 7一样，我们使用五种颜色来显示五个不同的分布，这是由于将 9915项测试重复五次所致。图1 的右侧展示了 40 个训练数据位置和这些 9915 个留出邮政编码。

6 讨论

6.1 发现的改进类型

6.1.1 OLC。在一次运行中，Magpie 找到了一个补丁，在十个训练位置上使 L1 数据缓存未命中减少了 8%。它只是删除了一行代码。被删除的行属于 OLC 的命令行验证部分。由于所有测试都使用格式正确的命令行，因此用于检查命令行解析错误的检查从未触发。因此，被删除的行从未执行。Magpie 报告的 8% 改进似乎只是 perf 测量中的噪声。也就是说（与下一节的 H3 一样），该补丁并未持续降低 L1 数据缓存未命中。不过，正如

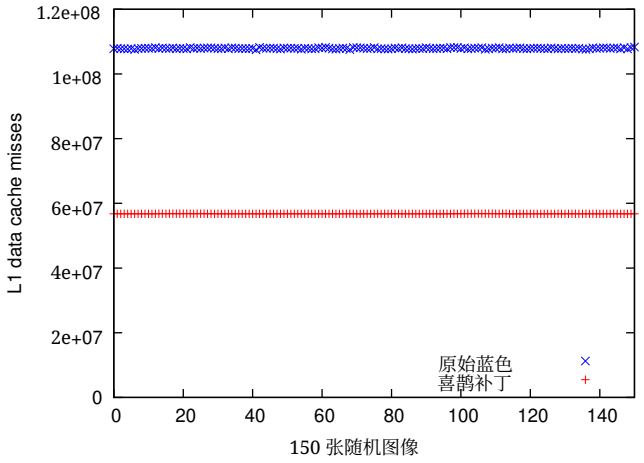

图 9：Magpie“Blue”补丁在 150 张留出图像上的样本外性能。（图 2 包含一张训练图像。）所有情况下，补丁返回的蓝色像素计数都与原始代码相同，但 L1 数据缓存未命中数仅约为原来的一半。

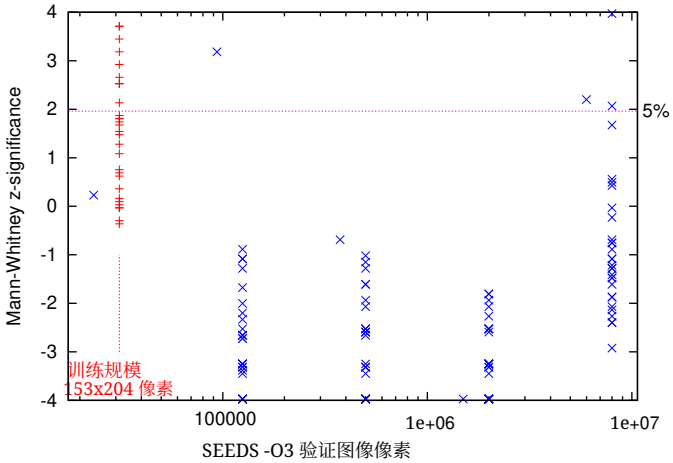

图 10：Magpie SEEDS 补丁在 150 张不同尺寸留出图像（例如图 4）上的样本外性能。所有情况下，补丁对图像的分割结果都与 OpenCV 完全相同。即使重复测量 11 次，L1 缓存的测量结果仍然非常嘈杂。尽管如此，对于与训练图像（图 3）尺寸相同的 153×204 图像（红色 +），补丁往往会减少 L1 数据未命中；而对于更大的图像（蓝色 ×），表现则更差。注意 x 轴采用对数刻度。

正如预期，它在功能上具有泛化能力，并通过了全部 10 000 个留出的邮政地址测试（图 1 左侧以蓝色 + 显示）。

6.1.2 H3. 在一次 H3 运行中，找到的最佳补丁包含 12 处更改。其中两处是行替换，九处是行删除，还有一处是行插入，共计 14 处行更改，而且全部不影响功能。其中两处删除了从未使用过的代码行。一处包含一个头文件，但由于条件编译，该文件没有任何影响。另有两处添加了

常量数组中的代码行，但这些数组从未被使用。九处删除了常量数组中的数据；其中只有两种情况下数组会被引用，而且在所有情况下，只会使用数组中未更改的部分。正如预期的那样，打过补丁的代码通过了全部 10000 个验证用例。

6.1.3 SEEDS. 在一次运行中，Magpie 找到了一个包含七处改动的补丁，这些改动使报告的 L1 数据缓存未命中总数减少了 7.4%：

删除第 254 行，删除第 204 行，在第 105 行的第 102 个位置之前插入，删除第 247 行，在第 66 行的第 87 个位置之前插入，删除第 63 行，在第 109 行的第 137 个位置之前插入。

在三个位置，GI 利用了运行时可用、但 GCC -O3 编译器无法获得的信息：int seeds_prior 始终为 2。因此，if(seeds_prior) 这一行始终为真，可以删除；而 C++ switch(seeds_prior) 语句始终会进入 case 2:。因此，删除 case 1: 和 case 5: 不会改变程序的语义。事实上，case 3: 和 case 4: 语句也同样可以删除。实际上，在其他运行中，Magpie 删除了 case 3: 和/或 case 4:。（这些冗余的 case 语句位于第 247–249 行和第 254 行。）

这些可以视为传统的 GI 加速改动，即移除不必要的操作。它们也可能对数据缓存产生潜在的有利影响，不过只有删除第 63 行的 if(seeds_prior) 会直接影响数据使用情况。缩减代码大小可能会通过改变优化编译器生成的机器码，对数据缓存产生微妙的二阶影响。

删除第 204 行则会产生相反的效果，因为它移除了 if(labelA != labelB)，而这个条件很少成立。因此，该补丁会导致下一行的 update() 被更频繁地调用。但每次调用都会为像素 (w-1,y) 计算出相同的标签，因此不会改变图像的分割结果。在适应度函数中不存在取舍：该补丁可以随意浪费 CPU 指令，重要的只是减少 L1 数据缓存未命中。第 204 行之后的 update() 会在一个包含 153 次循环迭代的循环中调用；在下一次循环迭代中，需要访问与 (w-1,y) 相邻的数据，即 (0,y_{+1})。因此，不必要的 update() 调用可能有助于将数据保留在 L1 缓存中[5]。这一论点应该适用于任何尺寸的图像，但图 3 表明，图像尺寸每扩大四倍时，对 L1 数据缓存的整体益处并非每次都能保持。这可能是因为图像越大，update() 使用的其他数组就越多，从而抵消了试图将标签像素数据保留在 L1 缓存中的益处。

另外三处修改插入了代码行的副本，这些代码同样不会改变数值，但会导致它们被重新计算：

priorB = threebyfour(x, y, labelB);（第 87 行之前）int a11 = Labels((y - 1) * width + (x -1));（第 102行）int a23 = Labels((y)*width + (x + 1));（第 137 行）编译器识别出 inta11和 int a23 未被使用，但这并不能确保它会忽略它们，因此它可能会生成代码来访问像素 (x-1,y-1)和(x+1,y) 的标签。两者都出现在一个以错误顺序扫描图像的双重嵌套循环中：也就是说，内层循环采样的数据相距很远，而不是访问相邻的缓存行。因此，可用的 512 个L1缓存行可能会承受很大压力。对于添加int a11 的补丁来说，标签 (x-1,y-1) 似乎通常几乎会立即被需要，所以重复计算int a11 可能几乎不会增加缓存使用。而 int a23补丁读取的是像素 (x+1,y) 的标签，该标签最近刚被读取过，因此再次重新计算int a23 可能同样几乎不会对缓存造成影响。嵌套循环使分析变得困难，而且可能存在重要的数据局部性影响，类似于前文中针对 update() 所提出的影响。在较大的图像中，这些影响可能会完全压垮计算机的 512 条 L1 数据缓存行。

相比之下，为设置 priorB 而在第 87 行之前插入的额外调用 threebyfour(x, y, labelB); 位于一对嵌套循环中，这些循环按“正确”的顺序访问像素数据，也就是沿着 x 轴访问。如果调用，threebyfour 会读取三行中四个相邻的像素标签。这些数据会占用三到六条缓存行（具体取决于 x 维度如何跨越 64 字节边界）。与上文讨论的第 204 行补丁类似，该补丁会导致 update() 被调用更多次；考虑到这些嵌套循环以及 threebyfour 刚刚被调用过，额外调用 threebyfour 可能有助于将数据（这些数据已经位于 L1 缓存中）保留在其中。紧邻这些 3×4 像素的数据，因此也位于相同缓存行中的数据，几乎肯定会在下一次循环迭代中立即用到。

6.2 非平稳噪声、性能分析、变异及其他搜索算子

如第 2.2 节所述，鉴于运行时间测量中经常出现较大的正向离群值，我们在适应度函数中采用了第一四分位数，因为它是一种稳健统计量，相对不易受到正向离群值的影响。尽管图 7 和图 8 未绘出少量较大的正向离群值，但它们表明，缓存未命中的分布噪声很大，而且比预期更加对称，近似高斯分布。鉴于这种明显的对称性，中位数或许比第一四分位数更适合作为一致的适应度度量（另见图 11）。

然而，图 7 和图 8 中以不同颜色显示的多模态分布揭示了另一个鲜少讨论的问题：L1D 噪声并非平稳的 [34]，而是受到某种未知漂移的影响。经典论述假定多次测量相互独立且同分布（IID），据此认为增加测量次数 n 会按 $\sqrt{n}$ 的比例降低噪声的影响。然而，这忽略了这样一个事实：如果噪声是非平稳的，那么在 Magpie 运行期间（实际上，在任何进化计算（EC）运行期间）进行的测量都会发生漂移。这种漂移不仅发生在 EC 运行期间，在使用进化所得产物时也会发生。我们或许不仅需要在每次适应度评估中进行多次 L1D 测量，还需要在 EC 运行期间估算漂移。也许可以通过运行一些已知的固定示例代码来实现。如果在线漂移估算证明有效，那么在 EC 运行期间投入精力应对噪声很可能是值得的，因为这应能带来更好、更稳健的解决方案。

Magpie 已针对可能被调用的整个函数进行优化。但在我们最大的示例 H3 中，这还不够。因为其中既有从未执行过的代码行，也有从未使用过的预设数据值，Magpie 却在浪费精力尝试优化它们。在 GI 中，通常会先对待进化的程序进行性能分析，然后只针对确实执行过的代码 [24]。在这些示例中，

[5]缓存行大小为 64 字节，因此可以容纳 16 个水平相邻像素的标签。

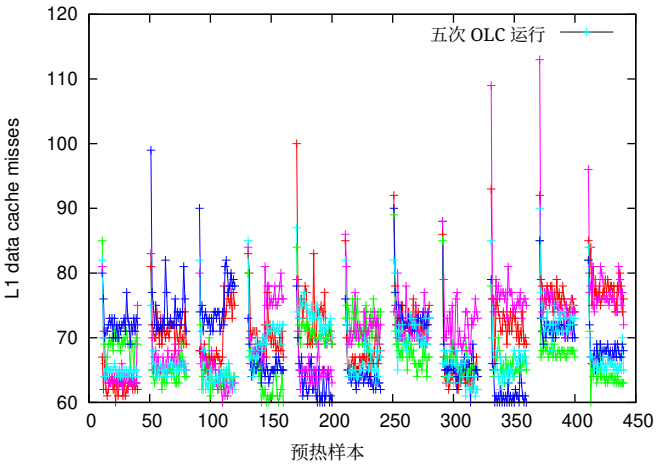

图 11：Magpie 的 OLC 预热过程中的 L1 数据缓存未命中（即反复测量同一程序时）。请注意：存在离群值；一组测量中的首次测量往往远大于其余测量；同一软件在同一台计算机上的不同运行之间，既有系统性差异，也有随机差异。

未使用性能分析。事实上，H3 中存在大量未使用数据，这暗示着还存在一个进一步的问题。早期的性能分析主要集中于代码执行。看来当时并未考虑到，在日常平凡操作期间，代码可能会创建一些并未被使用的数据（因此需要对其进行维护）。

当然，Magpie 还有更复杂的、能感知语法的处理方法，而且这些方法也可能从配置文件数据中获益。显然，较新的 Magpie 版本已经会排除一些简单补丁，例如移动 #include 文件，或仅仅交换只包含一个右花括号“}”的行。这些方法或许有帮助，但我们担心，一如既往，由适应度驱动的进化会找到利用代码更改的办法，通过作出其他尚未想到的、看似“无用”的更改来达到目的。

通过维护一个禁忌列表来强制每个补丁产生新的语义变化（正如SEEDS所采用的方式），无法解决这样的问题：大量无谓的工作被耗费在生成补丁上，而这些补丁要么改变未使用的代码，要么改变未使用的数据。它用来防止重复的禁忌列表，是基于编译器创建的目标文件的。对未使用代码或未使用预设数据的修改都会改变目标文件，因此，尽管这些修改毫无用处，却仍会显得像是合理的语义变化。对于 Magpie的局部搜索而言，这个问题更加严重，因为由于适应度函数的噪声，对未执行代码或未使用数据进行的看似“无用”修改，可能会显得有益，从而将搜索引向无效的方向。然而，我们需要保持谨慎，因为缓存的行为通常是专有的，即使是“显然无用”的修改，其确切影响在实践中也无法得知。例如，即使 GCC 编译器（使用 -O3）发出警告，称补丁引入了一个“未使用的变量”，它仍可能改变其生成的机器代码。因此，缓存的行为可能会发生改变。

到目前为止，借助Magpie，我们只使用了几种变异类型：行删除6、行插入和行替换。还有许多其他类型也是可行的。同样，我们只使用了 Magpie的局部搜索，还可以探索其他策略。事实上，Magpie 已经支持遗传编程。未来，Magpie 的参数搜索也可能应用于源代码之外的方面，例如编译和链接。

7 结论

我们采用了一个以 Python 编写的新型开源遗传改进工具 Magpie，并将其应用于来自 Google 和 Uber 的工业级 C 源代码、“Blue”图像处理问题，以及 OpenCV 最先进的图像分割 C++ 代码。具体而言，我们首次尝试优化硬件数据缓存层级的最深层级，而这对现代计算机至关重要。对于 OpenCV 的 SEEDS，在编译器 -O3 优化代码的基础上实现了 1% 的降低；而“Blue”的 L1 数据缓存未命中几乎减半（47%）。

回顾来看，或许期待搜索能够在我们的两个工业地理定位示例（OLC和 H3）中对数据缓存产生重大影响，是有些过于乐观了。毕竟，它们包含大量代码，而数据输入却非常少（两个浮点数）。也许更重要的是，一次典型运行只会产生几百次 L1 数据缓存未命中，因此可改进的空间很小。尽管如此，令人欣慰的是，在一分钟或几小时[7] 内，我们便找到了针对编译器优化代码的改进，而且这些改进至少在功能上具有普适性。

致谢

感谢 H.Wierstorf 在 gnuplot 方面提供的帮助。本研究得到 EP/P023991/1 和 Meta OOPS 项目的支持。

参考文献

[1] Nadia Alshahwan。2019。Facebook 中遗传改进的工业实践。在 *GI-2019*，*ICSE* 研讨会论文集 中，Justyna Petke、Shin Hwei Tan、WilliamB. Langdon和 Westley Weimer（编）编。IEEE，蒙特利尔，1。 http: //dx.doi.org/10.1109/GI.2019.00010 特邀主题演讲。 [2] Gabin An、Aymeric Blot、JustynaPetke和Shin Yoo。2019。PyGGI 2.0：与语言无关的遗传改进框架。在 第 *27* 届欧洲软件工程会议与软件工程基础研讨会联合会议论文集（*ESEC/FSE 2019*） 中，Sven Apel和Alessan- dra Russo（编）编。ACM，爱沙尼亚塔林，1100–1104。http://dx.doi.org/10.1145/ 3338906.3341184[3] Gabin An、JinhanKim、SeongminLee 和 ShinYoo。2017。PyGGI：用于遗传改进的 Python通用框架。在韩国软件大会论文集（*KSC 2017*） 中。韩国釜山，536–538。 https://coinse.kaist.ac.kr/publications/pdfs/An2017aa.pdf [4] Gabin An、Jinhan Kim和Shin Yoo。2018。使用 PyGGI 比较程序修复的行粒度和 AST 粒度级别。在 *GI-2018*，*ICSE* 研讨会论文集 中，Justyna Petke、Kathryn Stolee、WilliamB.Langdon和 Westley Weimer（编）编。ACM，瑞典哥德堡，
19 26
10 1145 3194810 3194814 – 。
[5]
1998
http://dx.doi.org/ 。/ 。Wolfgang Banzhaf、Peter Nordin、Robert E. Keller 和 FrankD. Francone。 。遗传编程——简介：计算机程序的自动进化及其应用。Morgan Kaufmann，美国加利福尼亚州旧金山。
h
t
t
p
s://
www.amazon.co.uk/Genetic-Programming-Introduction-Artificial-Intelligence/dp/155860510X
[6] AymericBlot和 Justyna Petke。2021。软件遗传改进的搜索启发式方法实证比较。*IEEE* 进化计算汇刊 25，5（2021 年 10 月），1001–1011。http://dx.doi.org/10.1109/TEVC.2021.3070271。

[6]删除是程序员加快代码运行速度最常用的方法 [9]。[7]Magpie 使用 GNU C/C++ 编译器（-O3），每秒处理 0.7 到 10 个补丁，具体取决于 OLC、H3、Blue 或 SEEDS。相比之下，在我们先前关于 OLC 和 H3 的工作中 [21]，使用 Clang 14.0.0 每秒处理 0.25 到 1.5 个 LLVM IR 补丁，具体取决于实验，以及是否使用 -O3。

[7] Aymeric Blot 和 Justyna Petke。2022年。MAGPIE：通过软件进化实现机器自动化通用性能改进。arXiv。http://dx.doi.org/ 10.48550/ARXIV.2208.02811[8] Alexander E. I. Brownlee、Justyna Petke、Brad Alexander、EarlT. Barr、Markus Wagner和 David R. White。2019年。GIN：让遗传改进研究变得简单。在 *GECCO '19*，Manuel Lopez-Ibanez、Thomas Stuetzle、Anne Auger、Petr Posik、LesliePeprez Caceres、Andrew M.Sutton、Nadarajen Veerapen、Chris-tine Solnon、Andries Engelbrecht、Stephane Doncieux、Sebastian Risi、Penousal Machado、Vanessa Volz、ChristianBlum、Francisco Chicano、Bing Xue、Jean-Baptiste Mouret、Arnaud Liefooghe、Jonathan Fieldsend、Jose Antonio Lozano、Dirk Arnold、Gabriela Ochoa、Tian-Li Yu、Holger Hoos、Yaochu Jin、Ting Hu、Miguel Nicolau、Robin Purshouse、ThomasBaeck、Justyna Petke、Giuliano An- toniol、Johannes Lengler 和PerKristian Lehre（编）。ACM，捷克共和国布拉格，985 993 10 1145 3321707 3321841 – 。[9] http://dx.doi.org/. / 。James Callan、Oliver Krauss、Justyna Petke 和 FedericaSarro。2022 年。Android开发者如何改进软件的非功能属性？实证软件工程 27（2022），文章 113。http://dx.doi.org/10.1007/s10664- 022-10137-2 专题合集：软件性能。[10] Jack J. Dongarra。2022 年。数学软件的进化。*ACM* 通讯 65, 12 2022 66 72 10 1145 3554977 （12 月），– 。[11] http://dx.doi.org/ . / WilliamFeller。 。概率论及其应用导论 （第 2版）。第1 卷。John Wiley and Sons，纽约。[12] Stephanie Forrest、ThanhVu Nguyen、Westley Weimer 和Claire Le Goues。2009 年。一种自动化软件修复的遗传编程方法。在*GECCO'09*，Guenther Raidl、Franz Rothlauf、Giovanni Squillero、Rolf Drechsler、Thomas Stuetzle、Mauro Birattari、ClareBates Congdon、Martin Middendorf、Christian Blum、Carlos Cotta、Peter Bosman、Joern Grahl、Joshua Knowles、David Corne、Hans-Georg Beyer、Ken Stanley、Julian F. Miller、Jano van Hemert、Tom Lenaerts、Marc Ebner、Jaume Bacardit、Michael O'Neill、Massimiliano Di Penta、Benjamin Doerr、Thomas Jansen、RiccardoPoli 和Enrique Alba（编）。ACM，加拿大蒙特利尔，947–954。http://dx.doi.org/10.1145/1569901.1570031GECCO 2019 十年最具影响力论文奖，最佳论文。[13] Gordon Fraser和Andrea Arcuri。2011年。EvoSuite：面向对象软件的自动化测试套件生成。在 8th 欧洲软件工程会议暨 *ACM SIGSOFT*软件工程基础研讨会（*ESEC/FSE'11*）。ACM，匈牙利塞格德，416–419。http://dx.doi.org/10.1145/2025113.2025179[14] Mark Harman、WilliamB.Langdon和Westley Weimer。2013年。用于逆向工程的遗传编程。在 第 *20* 届逆向工程工作会议（*WCRE 2013*），RoccoOliveto和 Romain Robbes（编）。IEEE，德国科布伦茨，1–10。http://dx.doi.org/10.1109/WCRE.2013.6671274 特邀主题演讲。[15] LinseyKitt 和 Myra B. Cohen。2021 年。程序修复的部分规范。在 *GI @ ICSE 2021*，Justyna Petke、Bobby R. Bruce、Yu Huang、Aymeric Blot、Westley Weimer 和W. B. Langdon（编）。IEEE，互联网，19–20。http://dx.doi.org/10.1109/GI52543.2021.00012[16] JohnR.Koza。1992 年。遗传编程：通过自然选择进行计算机编程。MITPress，美国马萨诸塞州剑桥。http://mitpress.mit.edu/books/genetic-programming [17] John R. Koza、Forrest H BennettIII 和 Oscar Stiffelman。1999 年。遗传编程作为达尔文式发明机器。在 遗传编程，*EuroGP'99* 论文集（*LNCS*，第 *1598* 卷），Riccardo Poli、Peter Nordin、William B.Lang- don 和 TerenceC.Fogarty（编）。Springer-Verlag，瑞典哥德堡，93–108。http://dx.doi.org/ 10 1007 3 540 48885 5_8 。 /--- OliverKrauss。 。Amaru[18]——一个将遗传改进与模式挖掘相结合的框架。在 *GECCO 2022*，Bobby R.Bruce、Vesna Nowack、Aymeric Blot、Emily Winter、W. B. Langdon 和 Justyna Petke（编）。计算机协会，美国波士顿，1930–1937。http://dx.doi.org/10.1145/ 3520304.3534016[19] W. B.Langdon。2018 年。遗传改进 *GISMOE Blue* 软件工具演示。技术报告 RN/18/06。伦敦大学学院，英国伦敦。http: // www.cs.ucl.ac.uk/fileadmin/user_upload/blue.pdf[20] W. B. Langdon。2020 年。遗传编程的遗传改进。在 *GI @ CEC 2020* 特别会议，Alexander (Sandy) Brownlee、Saemundur O. Haraldsson、Justyna Petke和John R. Woodward（编）。IEEE 计算智能学会，IEEE Press，互联网，论文编号 id24061。http://dx.doi.org/10.1109/CEC48606. 2020.9185771[21] William B. Langdon、Afnan Al-Subaihin、Aymeric Blot和David Clark。2023 年。LLVM 中间表示的遗传改进。在 *EuroGP 2023*：第 *26* 届欧洲遗传编程会议论文集（*LNCS*，第 *13986* 卷），Gisele Pappa、Mario Giacobini 和 Zdenek Vasicek（编）。Springer Verlag，捷克共和国布尔诺，244–259。http://dx.doi.org/10.1007/978-3-031- 29573-7_16 即将出版。[22] William B. Langdon 和 Bradley J. Alexander。2023 年。使用 Magpie 对 OLC 和 H3 进行遗传改进。在 第 *12* 届遗传改进国际研讨会 *@ICSE 2023*，Vesna Nowack 、Markus Wagner、GabinAn、Aymeric Blot和 Justyna Petke（编）。IEEE，澳大利亚墨尔本。http://www.cs.ucl.ac.uk/staff/ W.Langdon/ftp/papers/langdon _2023_GI.pdf 已接收。[23] W. B. Langdon 和 M. Harman。2010。基于 nVidia 模板进化出 CUDA 内核。载于 *2010 IEEE* 世界计算智能大会，Pilar So- brevilla（编）。IEEE，巴塞罗那，2376–2383。http://dx.doi.org/10.1109/CEC.2010. 5585922[24] William B. Langdon 和 Mark Har man。2015。使用遗传编程优化现有软件。*IEEE* 进化计算汇刊 19，1（Feb. 2015 118 135 10 1109 2013 2281544 ），–。http://dx.doi.org/ . /[25] TEVC. . William B. Langdon、Brian Yee Hong Lam、Justyna Petke 和 Mark Harman。2015。使用遗传编程改进 CUDA DNA 分析软件。载于*GECCO '15*，Sara Silva、Anna I Esparcia-Alcazar、Manuel Lopez-Ibanez、Sanaz Mostaghim、Jon Timmis、Christine Zarges、Luis Correia、Terence Soule、Mario Giacobini、Ryan Urbanowicz、Youhei Akimoto、Tobias Glasmachers、Francisco Fernandez de Vega、Amy Hoover、Pedro Larranaga、Marta Soto、Carlos Cotta、Francisco B. Pereira、Julia Handl、Jan Koutnik、Antonio Gaspar-Cunha、Heike Trautmann、Jean-Baptiste Mouret、Sebastian Risi、Ernesto Costa、Oliver Schuetze、Krzysztof Krawiec、Alberto Moraglio、Julian F. Miller、Pawel Widera、Stefano Cagnoni、JJ Merelo、Emma Hart、Leonardo Trujillo、Marouane Kessentini、Gabriela Ochoa、Francisco Chicano 和 Carola Doerr（编）。ACM，马德里，1063–1070。http://dx.doi.org/ 10 1145 2739480 2754652 . / . William B. Langdon、Justyna Petke 和 Ronny Lorenz[26]。2018。进化出更优的 RNAfold 结构预测。载于*EuroGP 2018*：第 *21* 届欧洲遗传编程会议论文集（*LNCS*，第 *10781* 卷），Mauro Castelli、Lukas Sekanina 和 Mengjie Zhang（编）。Springer Verlag，意大利帕尔马，220–236。http://dx.doi.org/ 10 1007 978 3 319 77553 1_14 . / - - - - William B. Langdon、David R. White、Mark Harman、Yue Jia 和 Justyna Petke[27]。2016。受应用程序接口约束的遗传改进。载于第八届基于搜索的软件工程国际研讨会论文集，*SSBSE 2016*（*LNCS*，第 *9962* 卷），Federica Sarro 和 Kalyanmoy Deb（编）。Springer，美国北卡罗来纳州罗利，224 230 10 1007 978 3 319 47106 8_16 –。2019 http://dx.doi.org/[28] . / - - - - Claire Le Goues、Michael Pradel 和 Abhik Roychoudhury。. 自动程序修复。*ACM* 通讯 62，12（Dec. 2019），56–65。http://dx.doi.org/10.1145/3318162[29] Pierrick Legrand、Claire Bourgeois-Republique、Vincent Pean、Esther Harboun-Cohen、Jacques Levy-Vehel、Bruno Frachet、Evelyne Lutton 和 Pierre Collet。2007。人工耳蜗调试的交互式进化。遗传编程与可进化机器 8，4（Dec. 2007），319–354。http://dx.doi.org/10.1007/s10710- 007-9048-4 遗传与进化计算医学应用特刊。[30] Sherlock A. Licorish 和 Markus Wagner。2022。论结合 GIN 与 PMD 改进 Stack Overflow 代码片段的效用。ArXiv。https: // dblp.org/rec/journals/corr/abs-2202-01490.bib [31] Ibrahim Mesecan、Michael C. Gerten、James I. Lathrop、Myra B. Cohen 和 Tomas Haddad Caldas。2021。CRNRepair：化学反应网络的自动程序修复。载于*GI @ ICSE 2021*，Justyna Petke、Bobby R. Bruce、Yu Huang、Aymeric Blot、Westley Weimer 和 W. B. Langdon（编）。I EEE，互联网，23–30。http://dx.doi.org/10.1109/GI52543.2021.00014 最佳论文奖。[32] Martin Monperrus。2018。自动软件修复：参考文献目录。计算机调查 51, 1 2018 17 10 1145 3105906 （Jan. ），文章编号 . http://dx.doi.org/ . / Gordon E. Moore[33]。. 在集成电路上集成更多元件。*Electronics* 38，8（1965 年 4 月 19 日），114–117。[34] David Moskowitz。2016。遗传编程中用于改进非平稳、非线性时间序列预测的自动定义模板。博士论文。Nova Southeastern University 工程与计算学院，美国。http://nsuworks.nova.edu/gscis_etd/953/ [35] Mike Papadakis、Yue Jia、Mark Harman 和 Yves Le Traon。2015。平凡编译器等价性：对一种简单、快速且有效的等价变异体检测技术的大规模实证研究。载于第 *37* 届软件工程国际会议（*ICSE 2015*）。佛罗伦萨。http://pages.cs.aueb.gr/~mpapad/papers/ ICSE15B.pdf 即将发表。[36] Justyna Petke 和 Aymeric Blot。2020。改进基于测试的程序修复中的适应度函数。载于首届自动程序修复国际研讨会（*APR@ICSE 2020*），Shin Hwei Tan、Sergey Mechtaev、Martin Monperrus 和 Mukul Prasad（编）。计算机协会，互联网，13–14。http://dx.doi.org/ 10 1145 3387940 3392180 . / . Justyna Petke 和 Alexander Brownlee[37]。2019。使用 Gin 改进软件：案例研究。载于*SSBSE 2019*（*LNCS*，第 *11664* 卷），Shiva Nejati 和 Gregory Gay（编）。Springer，爱沙尼亚塔林，183–189。http://dx.doi.org/10.1007/978-3-030- 27455-9 14[38] Justyna Petke、Saemundur O. Haraldsson、Mark Harman、William B. Langdon、David R. White 和 John R. Woodward。2018。软件遗传改进：综合综述。*IEEE* 进化计算汇刊 22，3（June 2018 415 432 10 1109 2017 2693219 ），–。2008 http://dx.doi.org/doi: . /TEVC. . Riccardo Poli、William B. Langdon[39] 和 Nicholas Freitag McPhee。. 遗传编程指南。通过 http://lulu.com 出版，并可在 http://www.gp-field-guide.org.uk 免费获取。http://www.gp-field-guide.org. uk（J. R. Koza 参与撰写）。

[40] Conor Ryan。1999。使用遗传编程自动重构软件。遗传编程，第 2 卷。Kluwer Academic Publishers。http://www.amazon.com/exec/obidos/ASIN/0792386531/qid%3D943291341/ 102-9266197-5591202[41] Marta Smigielska、Aymeric Blot 和 Justyna Petke。2021。遗传改进中的统一编辑选择：变异算子效能的实证分析。载于*GI @ ICSE 2021*，Justyna Petke、Bobby R. Bruce、Yu Huang、Aymeric Blot、Westley Weimer 和 W. B. Langdon（编）。IEEE，互联网，1–8。http://dx.doi.org/10.1109/ GI52543.2021.00009 [42] Lee Spector、W. B. Langdon、Una-May O'Reilly 和 Peter J. Angeline（编）。1999。遗传编程进展 *3*。MIT Press，美国马萨诸塞州剑桥。http://dx.doi.org/ 10 7551 1110 001 0001 . /mitpress/ . . Leonardo Vanneschi 和 [43] Sara Silva。. 智能系统讲座。Springer。http://dx.doi.org/ 10 1007 978 3 031 17922 8 . / - - - - Westley Weimer、Stephanie [44] Forrest、Claire Le Goues 和 ThanhVu Nguyen。2010。基于进化计算的自动程序修复。*ACM* 通讯 53，5 （ 2010 109 116 10 1145 1735223 1735249 年6月），–。 [45] http://dx.doi.org/。/。Westley Weimer、ThanhVu Nguyen、Claire LeGoues 和 Stephanie Forrest。2009。使用遗传编程自动寻找补丁。载于*2009*年国际软件工程大会（*ICSE*），Stephen Fickas（编）。温哥华， 364 374 10 1109 2009 5070536 –。 [46] 2017 *2017*, http://dx.doi.org/。/ICSE.。David R. White。。无需耗时的遗传改进。载于*GI*-Justyna Petke、David R. White、W. B. Langdon 和 Westley Weimer（编）。ACM，柏林，1549–1550。 http: //dx.doi.org/doi: 10 1145 3067695 3082515 . /。Shengjie Zuo、Aymeric [47] Blot 和 Justyna Petke。2022。遗传改进工具在改善软件非功能属性方面的评估。载于*GECCO 2022*，Bobby R. Bruce、Vesna Nowack、Aymeric Blot、Emily Winter、W. B. Langdon 和 Justyna Petke（编）。计算机协会，波士顿，美国，1956–1965。 http://dx.doi.org/10.1145/3520304.3534004 最佳论文奖。