arXiv:2301.08665v1 [cs.SE] 20 Jan 2023

MTGP：结合蜕变测试与遗传编程

Dominik Sobania、Martin Briesch、Philipp Röchner 和 Franz Rothlauf

约翰内斯·古腾堡大学，德国美因茨
{dsobania,briesch,proechne,rothlauf}@uni-mainz.de

摘要。 遗传编程是一种以程序合成表现出色而闻名的进化方法。然而，它尚未成熟到足以应用于现实世界的软件开发，因为通常需要大量训练用例，才能生成能够泛化到未见测试用例的程序。在实践中，训练用例必须由用户耗费大量精力手动标注，因此我们需要一种能够以较少训练用例检验程序行为的方法。蜕变测试不需要带标签的输入/输出示例。相反，它会多次执行程序：首先使用给定的（随机生成的）输入，然后使用与之相关的输入，以检查观测输出之间是否满足某些用户定义关系。在本研究中，我们提出了 MTGP，将蜕变测试与遗传编程相结合，并研究其性能以及所生成程序的泛化能力。此外，我们还分析了泛化能力如何随给定的带标签训练用例数量而变化。我们发现，在几乎所有研究配置中，将蜕变测试与带标签训练用例结合使用，比单独使用带标签训练用例能够获得更高的泛化率。因此，如果训练数据的标注成本高昂，我们建议研究人员在其系统中采用蜕变测试。

关键词： 程序合成 · 蜕变测试 · 遗传编程。

1 引言

遗传编程（GP） [7,21]是一种基于进化算法的方法，可在给定的编程语言中自动生成满足用户定义要求的程序。在基于遗传编程的程序合成中，这些规范通常以输入/输出示例的形式给出，用于定义生成的程序针对给定输入所应产生的预期输出，并在进化搜索过程中用作训练数据。

随着新型选择方法 [17,31],变异算子 [16] 和语法设计技术 [10],的引入，基于遗传编程的程序合成近年来取得了长足进展。最近的研究表明，基于遗传编程的程序合成方法在性能上甚至可以与最先进的神经网络方法相媲美 [26]。

遗憾的是，基于遗传编程的程序合成尚未成熟到足以应用于现实世界的软件开发，因为通常需要大量输入/输出示例（文献中经常使用多达 200 个 [13]），才能生成不仅适用于训练用例、还能泛化到此前未见过的测试用例的程序。然而在实践中，训练用例必须由用户手动标注，这既耗费大量精力，也十分费时。因此，有必要减少所需训练用例的数量，以尽量降低用户的手动工作量。然而，单纯减少训练用例的数量并不足够，因为规模较小的训练集很容易导致过拟合，平均而言会使生成的程序泛化能力较差。因此，我们需要一种补充方法，在不增加更多人工定义的训练用例的情况下，指定并检查所需的程序行为。

采用蜕变测试 [6],时，我们无需额外手动标注训练用例，因为我们会多次执行程序（从一个随机输入开始，随后输入相关输入），并检查观测输出之间的关系在逻辑上是否符合用户的领域知识。例如，根据考试成绩评定等级的函数，可以用一个随机成绩来执行。如果随后提高该成绩，函数就必须返回相同或更高的等级；否则就违反了蜕变关系，该函数也就不正确。此类无需标注的蜕变关系，可以与传统的（但规模较小的）手动标注训练集结合使用。我们预计，这些额外的关系有助于提高遗传编程生成程序的泛化能力。

因此，在这项工作中，我们提出一种结合蜕变测试与遗传编程（MTGP）的方法，并研究其在一组常见程序合成基准问题上的性能，以及生成程序的泛化能力。为分析遗传编程的泛化能力如何取决于给定训练用例的数量，我们针对不同规模的（带标签）训练集开展实验。

MTGP 基于一种语法引导的遗传编程方法，该方法在进化过程中使用 lexicase [31]进行个体选择。由于词例选择并非基于聚合适应度值，而是考量程序在各个独立用例上的表现，因此非常适合在选择过程中同时考虑人工标注的训练用例和蜕变关系。为研究这种组合，我们分析了不同规模的人工标注训练集，并根据针对所考虑基准问题所定义的蜕变关系添加更多测试。具体而言，可根据随机输入自动构造基于蜕变关系的测试。首先在随机输入上运行候选程序，然后在后续输入（基于该随机输入）上运行。之后，根据预先定义的蜕变关系比较候选程序在随机输入和后续输入上的输出。如果关系成立，则测试通过；否则，测试失败。因此，蜕变测试的结果可以与基于带标签输入/输出示例的测试结果同等看待，区别在于，自动生成的蜕变测试用例无需耗费大量资源进行人工标注

。在我们的实验中发现，在几乎所有研究的配置中，将蜕变测试与人工标注的训练用例结合使用，比单独使用人工标注的训练用例（这是基于遗传编程的程序合成中的常见做法）能获得更高的泛化率。

在引言之后，我们将在第2节介绍近期与基于遗传编程的程序合成相关的工作，以及蜕变测试方面的研究。在第3节中，我们将详细介绍蜕变测试及其与遗传编程的结合。此外，我们还将介绍所使用的程序合成基准问题及其对应的蜕变关系。在第4节中，我们将介绍实验设置并讨论实验结果，最后在第5节总结全文。

2 相关工作

基于遗传编程的程序合成主要有两种方法：基于栈的遗传编程和语法引导的遗传编程 [30]。这些方法的主要区别在于程序表示，以及用于支持不同数据类型的技术。

基于栈的遗传编程方法使用不同的栈来分离不同的数据类型 [33]。在程序执行过程中，数据作为输入从相应的栈中取出，结果则被压回关联的栈中。在当前系统中，各条程序指令也各自存放在独立的栈上，这使得程序流程能够在运行时发生变化 [32]。

语法引导的遗传编程方法使用上下文无关文法来表示所支持的控制结构和语句之间的关系，并区分不同的数据类型 [11,34]。原则上，这项技术可用于创建任何编程语言的程序。然而，近年来，使用语法引导的遗传编程方法演化出的主要是 Python 程序 [10,24,27]。

无论采用哪种遗传编程方法，近年来程序合成的结果都有了显著改善，这主要归功于词例选择及其变体的应用 [10,12,15,17,19]。与锦标赛选择等选择方法不同，词例选择并不基于聚合适应度值（参见评估瓶颈 [22]），而是依据各个训练用例上的结果进行选择 [31]，这使得训练数据的结构也能够纳入考量。

无论采用哪种方法，基于遗传编程的程序合成主要使用输入/输出示例进行训练。不过，也有研究使用问题的文本描述[20] 或形式约束 [3] 等额外信息来提升程序合成的性能。

由于作为训练数据给出的输入/输出示例只是对问题的不完整定义，因此生成的程序不仅要能处理训练数据，还要能针对之前未见过的输入给出正确结果，这一点十分重要。为了提高遗传编程所生成程序的泛化能力，文献中提出了一些方法，例如通过后处理简化 [14] 生成更小的程序，或在运行结束时进行选择 [25]。另一种方法是使用批量词例选择 [1] 来提高泛化能力 [29]。此外，

最近还提出了一种方法，可用于预测遗传编程生成的程序能否泛化到未见过的数据 [28]。

Chen 等人提出的蜕变测试 [6] 是一种软件开发方法，无需明确指定测试的预期输出，即可检查被测程序的某些属性（详细说明见第3.1节）。在进化计算领域，蜕变测试曾被用于现有软件的遗传改进等场景 [23]。

然而，据我们所知，迄今尚无研究探讨将蜕变测试与遗传编程相结合对程序合成性能以及生成程序泛化能力的影响。

3 方法论

本节将介绍蜕变测试的基本原理，并通过一些示例说明其工作方式。此外，我们将介绍所选的程序合成基准问题及其蜕变关系。最后，我们将详细说明如何将蜕变测试与基于遗传编程的程序合成相结合。

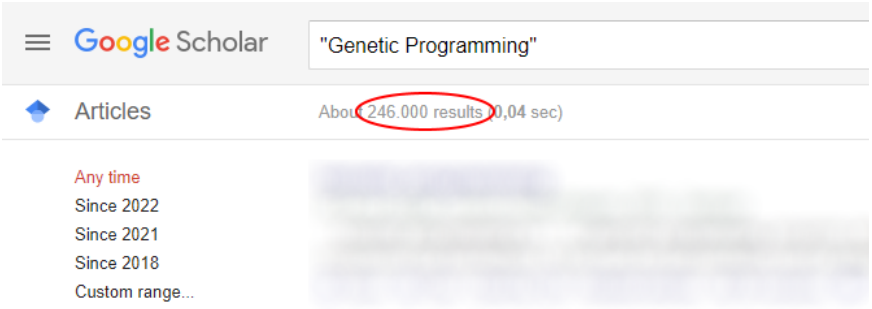
3.1 蜕变测试

蜕变测试 [6] 是一种软件开发方法，用于检查给定函数 f中的某些逻辑属性是否成立。这些属性由所谓的蜕变关系定义，用以描述给定的（随机）基础输入 I 及其观测输出 $f(I)$，与另一个后续输入 I' 及其相应输出 $f(I')$ 之间的逻辑联系。[1] 与传统测试方法相比，蜕变测试的主要优势在于，我们不需要昂贵的带标签的输入/输出示例，因为我们只关心观测输出 $f(I)$ 与 $f(I')$ 之间的蜕变关系是否成立。

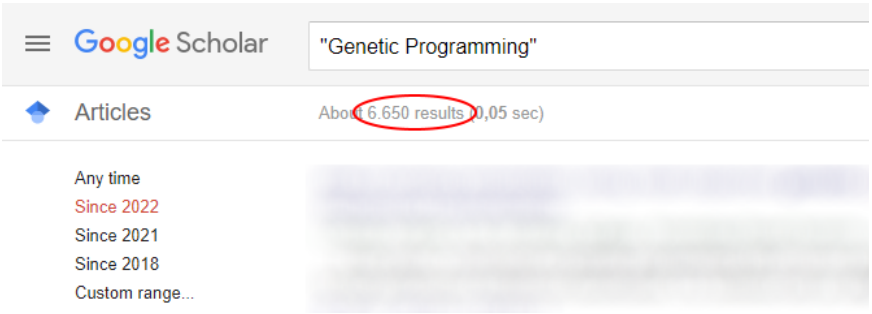
作为一个直观的例子（基于 [2] 中给出的示例），我们可以考虑一次简单的网络搜索。例如，作为基础输入 I，我们在科学搜索引擎中搜索确切词组“遗传编程”，不添加任何筛选条件，并得到 $f(I)$ 条结果。作为后续输入 I'，我们搜索相同的词组“遗传编程”，但将结果限制为 2022 年以来发表的文献，得到 $f(I')$ 条结果。我们将 $f(I) \geq f(I')$ 定义为蜕变关系，因为添加筛选条件后，结果数量应相同或更少。图 1 展示了 Google Scholar 的截图，以说明这一示例。我们看到 $f(I) = 246,000$，而 $f(I') = 6,650$，因此蜕变关系成立，因为 $f(I) \geq f(I')$。

作为第二个更偏技术性的例子，我们选择正弦函数，并预期它是 2π 周期的。因此，对于输入 I 和 $I' = I + 2\pi$,

[1] 也可以有多个后续测试，但在本研究中，我们只关注一个后续测试。



(a) 基础输入 I 的搜索结果（未启用筛选器）。



(b) 后续输入 I' 的搜索结果（已启用筛选器）。

图 1：Google Scholar 截图展示了一个简单的蜕变测试示例。对于基础输入 I (a)，我们看到相应的输出$f(I) = 246,000$；对于后续输入 I'(b)，输出为 $f(I') = 6,650$，因此定义的关系 $f(I) \geq f(I')$ 成立。

可以将蜕变关系定义为 $f(I) = f(I')$ [4,5]。图 2 展示了这个例子中的 $I = -3.7$。我们看到，构造的后续输入 $I' = -3.7 + 2\pi$ 会得到相同的结果，因此关系 $f(I) = f(I')$ 成立。

3.2 基准问题与蜕变关系

在评估中，我们从 Helmuth 和 Spector 的程序合成基准套件中选取了三个问题 [18]。根据近期的一项荟萃研究 [30]，，这些问题的难度各不相同，输入和输出所用的数据类型也各不相同。此外，为了便于比较，我们为每个基准问题定义了两种蜕变关系。不过，还可以构想出更多蜕变关系。我们重点关注相对简单的蜕变关系，即使用户仅具备有关该问题的基础领域知识，也能对其进行表述。基准问题和蜕变关系定义如下：

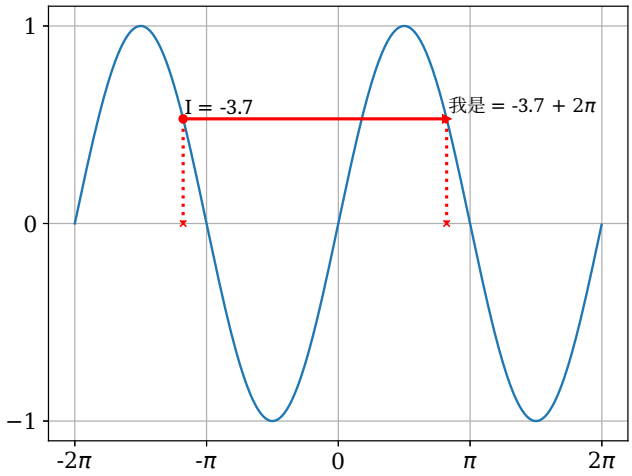


图 2：正弦函数示例。我们可以看到，基础输入$I = -3.7$ 和后续输入 $I' = -3.7 + 2\pi$ 的结果相同，因此满足定义的蜕变关系 $f(I) = f(I')$。

奇偶计数问题：

定义：应生成一个程序，用于返回给定整数列表中的奇数值数量。更形式化地说，我们要寻找一个函数 f，将给定的整数向量 $I = (x_1, ..., x_n) \in \mathbb{Z}^n$ 映射为奇数的数量$f(I) = c \in \mathbb{N}_0$，并将其作为返回值。

蜕变关系：1）作为第一个蜕变关系，我们要求当输入列表中增加任意数量的偶整数时，被测程序的输出不变。对于给定的输入 $I = (x_1, \ldots, x_n)$ 和程序 f，我们计算 $f(I) = c_I$。作为后续测试，我们通过在输入 I 中随机重复向量 (2, 4, 6)，构造扩展输入 $I' = (x_1, \ldots, x_n, 2, 4, 6, \ldots, 2, 4, 6)$。被修改输入所对应的输出为 $f(I') = c_{I'}$。当 $c_I = c_{I'}$时，该关系成立。我们选择 2、4 和 6，因为我们认为，具备该问题基础领域知识的用户都知道这些值是偶数（无需了解取模函数）。2）类似地，对于第二个蜕变关系，我们要求当输入中增加任意数量的奇整数时，输出值会增加。与第一个关系相反，我们通过随机重复若干次向量 (1, 3, 5)，将输入扩展为$I' = (x_1, \ldots, x_n, 1, 3, 5, \ldots, 1, 3, 5)$。因此，对应的输出为 $f(I') = c_{I'}$。当 $c_I < c_{I'}$时，该关系成立。这次我们选择 1、3 和 5 作为显而易见的奇数。

成绩等级问题：

定义：我们要寻找一个程序，根据给定的阈值将学生获得的数值分数映射为离散等级。具体而言，我们要寻找一个函数 f，将输入 I映射为其等级 $f(I) = g$，其中 $I = (t_1,\ t_2,\ t_3,\ t_4,\ s)$，且 $t_i,\ s \in \mathbb{N}_0$, $t_i,\ s \leq 100$对于 $i \in \{1,2,3,4\}$成立，$t_i < t_j$对于 $i > j$和$i,\ j \in \{1,2,3,4\}$成立；其中 t_i为阈值，s是学生获得的分数，$g = \{A,B,C,D,F\}$是对应的等级，其顺序为$A \succ B \succ C \succ D \succ F$。

蜕变关系：1）作为第一条关系，我们要求更高的数值分数对应相同或更高的离散等级。对于给定的有效随机输入 $I = (t_1, t_2, t_3, t_4, s)$ 和程序 f，我们计算 $f(I) = g_I$。之后，我们基于 I 创建一个经过修改的输入 $I' = (t_1, t_2, t_3, t_4, s + k)$，并使用 $k \in \{0, \ldots, 100 - s\}$，将等级 $f(I') = g_{I'}$ 作为后续测试。如果 $g_{I'} \succeq g_I$，则该蜕变关系成立。2）其次，我们定义相反的关系，要求较低的数值分数对应相同或更低的离散等级。我们基于 I 创建一个经过修改的输入 $I' = (t_1, t_2, t_3, t_4, s - k)$，并使用 $k \in \{0, \ldots, s\}$，将等级 $f(I') = g_{I'}$ 作为后续测试。如果 $g_{I'} \preceq g_I$，则第二条蜕变关系成立。

小或大问题：

定义：生成的程序应将给定的整数分类为小、大或介于两者之间。更准确地说，我们要寻找一个函数 f，将给定的整数 $I = n \in \mathbb{Z}$映射为其标签 $f(I) = l$，其中 $l = \{$“小”、“”、“大”$\}$的顺序为“小” $\prec$ “” $\prec$ “大”。如果 $n < 1,\ 000$，该函数 f应返回“小”；如果 $n \geq 2,\ 000$，则返回“大”；否则返回空字符串（“”）。

蜕变关系：***1***）首先，我们要求输入增大时，标签也必须根据定义的标签顺序保持不变或增大。对于给定的整数 $I = n$，我们计算得到的标签 $f(I) = l_I$。接着，我们将输入 $I' = n + k$与一个随机的 $k \in \mathbb{N}$一起进行变换，其对应的标签为 $f(I') = l_{I'}$。若 $l_{I'} \succeq l_I$，则该关系成立。***2***）因此，作为第二个关系，我们规定输入减小时，得到的标签必须根据标签顺序保持不变或减小。对于给定的输入 $I = n$，我们计算标签 $f(I) = l_I$。然后，我们将输入 $I' = n - k$与$k \in \mathbb{N}$一起进行变换，并得到 $f(I') = l_{I'}$。若 $l_{I'} \preceq l_I$，则该关系成立。

3.3 将 蜕变测试纳入 n GP

MTGP 得益于采用 lexicase 选择，因为 lexicase 会考虑单个用例，而不是聚合适应度值 [31]。因此，可以同时使用不同类型的（训练）用例进行选择。在 MTGP 中，这些用例包括人工标注的训练用例（其输入及对应输出已知），以及基于蜕变关系的用例。

对于人工标注的训练用例，输入和输出都是已知的。为了检查候选程序是否解决了某个训练用例，我们使用给定的输入 I 运行程序，并检查生成的输出 $f(I)$是否与预期输出O（在真实场景中由人工标注）相符。如果 $f(I) = O$，则测试通过，否则测试不通过。

为了测试为基准问题定义的蜕变关系，我们提供随机输入作为基础输入，而无需预先知道其输出。因此，可以低成本生成任意数量的随机条目。在我们的实验中，我们确保这些随机输入此前未出现在训练集或测试集中。实际应用中，可以自由选择这些输入。为了检查候选程序是否满足某个蜕变关系，我们使用给定的基础输入 I （随机生成的输入之一）运行程序，并保存输出 f（I）。随后，我们根据蜕变关系的操作规则随机更改输入，生成后续输入 I' ，并再次运行程序。如果观测输出 f（I）和 f（I'）满足该蜕变关系，则测试通过；否则测试失败。由于我们为实验中考察的所有基准问题都定义了两种蜕变关系，因此每次选定基础输入时，我们都会以概率 $p = 0.5$随机决定使用其中哪一种关系。

以这种方式使用蜕变测试的一个关键优势是，在一次遗传编程运行过程中，总会执行许多不同的蜕变测试，因为给定基本输入的变换是随机进行的。我们希望这能进一步提升生成程序的泛化能力。

4 实验与讨论

本节将研究 MTGP 的性能，并分析生成的程序对之前未见过的测试用例的泛化效果。为分析泛化能力如何取决于给定训练用例的数量，我们针对不同的带标签训练集大小开展实验。下文将介绍实验设置并讨论结果。

4.1 实验设置

在实现 MTGP 时，我们使用 PonyGE2 框架 [9]。我们所用的语法基于 PonyGE2 框架为基准套件 [18] 中的问题提供的程序综合语法，并遵循 Forstenlechner 等人 [10] 提出的原则：问题的语法仅限于该问题输入和输出所定义的数据类型（及依赖函数），以及必需的基本数据类型（例如整数和布尔值）。这样既能使所用语法保持精简、高效，又能保留足够的表达能力。

我们使用位置无关生长 [8] 初始化一次运行，并将种群规模设为 1，000。初始树深度的最大值设为 10，整体树深度的最大值限制为 17。变异算子使用子树交叉

和子树变异。子树交叉的概率设为 0.9，子树变异的变异步数设为 1。如上所述，我们使用 lexicase [31] 作为选择方法。若找到能够解决所有带标签训练样例和所有已定义蜕变测试的程序，或达到 300 代，AGP 运行即停止。

对于每个纳入研究的基准问题，我们都有 200 个带标签训练用例可用于实验。由 1,000 个带标签用例组成的测试集用于检验候选程序是否也能泛化到未见过的用例。此外，我们还提供了一组大型随机生成的输入（每个基准问题 800 个），并将其用作基础输入，以检验定义的蜕变关系对候选程序是否成立。在实验中，我们会根据下文所述的配置，从这些可用的训练用例和随机生成的输入中进行选择。

4.2 结果与讨论

为了研究在遗传编程中使用蜕变测试的影响，我们比较了两种方法：一种是仅使用训练数据的标准遗传编程方法，另一种是还纳入蜕变测试的新型 MTGP 方法。此外，我们还考察了标准遗传编程和 MTGP 在不同训练集规模下的表现 $|T_{\text{training}}|$。因此，

表1：标准遗传编程和 MTGP 在所有研究的带标签训练集大小 $|T_{\text{training}}|$及基准问题上的训练集 s_{training} 和测试集 s_{test} 成功率。对于 MTGP，我们还使用了 $200 - |T_{\text{training}}|$ 个蜕变测试。测试集 s_{test} 上的最高成功率以**粗体** 显示。

		标准遗传编程		**MTGP**	
问题	$\boldsymbol{\lvert T_{\text{training}}\rvert}$	$\boldsymbol{s_{\text{training}}}$	$\boldsymbol{s_{\text{test}}}$	$\boldsymbol{s_{\text{training}}}$	$\boldsymbol{s_{\text{test}}}$
计数奇偶	25	43	**28**	32	20
	50	59	**46**	43	35
	100	70	**64**	61	59
	200	83	81	-	-
等级	25	83	4	74	**14**
	50	86	12	76	**15**
	100	81	23	84	**33**
	200	93	45	-	-
小或大	25	94	2	58	**7**
	50	79	11	77	**25**
	100	91	**35**	86	29
	200	89	43	-	-

我们分析了两种方法在带标签训练集大小分别为 25、50、100 和 200 时的表现。MTGP 还会使用 $200 - |T_{\text{training}}|$ 个蜕变测试，使其在训练阶段恰好考虑 200 个测试（例如，对于 $|T_{\text{training}}| = 25$，MTGP 使用 175 个蜕变测试）。

首先，我们研究训练集 s_{training}和测试集 s_{test} 上的成功率，并将其作为性能指标。训练集 s_{training} 上的成功率衡量的是所有给定训练用例（MTGP 还包括蜕变测试）都能成功通过的程序被找到的运行次数所占的百分比。为了确定测试集 s_{test}上的成功率，我们从每次运行中选取在训练数据上表现最佳的候选程序，并计算能够解决所有此前未见测试用例的候选程序所占的百分比。对于每种研究配置，我们均运行了 100 次。

表1展示了标准遗传编程和 MTGP 在所有研究的带标签训练集大小 $|T_{\text{training}}|$（以及相应的蜕变测试）和基准问题上取得的训练集 s_{training} 及测试集 s_{test} 成功率。测试集 s_{test} 上的最佳成功率以**粗体** 显示。

对于大多数配置，标准遗传编程在训练数据 s_{training} 上的成功率高于 MTGP。我们推测，这是因为额外的蜕变测试避免了对训练数据过拟合。在测试数据上，对于成绩等级问题以及小或大问题的大多数配置，MTGP 的表现最佳（与相应的标准遗传编程运行相比）。对于奇偶计数问题，s_{test} 的最佳结果由标准遗传编程取得。

然而，比单纯的成功率更重要的是，基于训练数据找到的程序在未见过的测试用例上的泛化能力。在实践中，如果某种程序合成方法以泛化能力强而著称，那么首次运行未找到解时，只需再次运行即可。如果预期泛化能力较差，那么无论在训练数据上取得的成功率如何，都难以确定基于训练数据找到的解是否也适用于此前未见过的测试用例。在真实场景中，无法获得大量额外测试用例（用于检查泛化能力），因为手动标注额外的输入/输出示例成本过高。因此，在第二步中，我们分析泛化率

$$G = \frac{s_{\text{test}}}{s_{\text{training}}} \cdot 100.$$

表2展示了标准遗传编程和 MTGP 在所有研究的带标签训练集大小 $|T_{\text{training}}|$（以及相应的变异测试）和基准问题上的泛化率 G。同样，最佳泛化率 G 以**粗体** 字体显示。

我们发现，平均而言，MTGP 能够取得最佳泛化率 G；在标准遗传编程和 MTGP 都有结果的 9 种配置中，MTGP 有 7 种表现最佳。当仅使用较小的带标签训练集（$|T_{\text{training}}| = 25$）时，成绩等级问题和小或大问题的差异尤其明显。对于成绩等级问题，MTGP 的泛化

表2：标准遗传编程和 MTGP 在所有研究的带标签训练集大小 $|T_{\text{training}}|$ 及基准问题上的泛化率 G 。对于 MTGP，除了所考虑的带标签训练用例之外，我们还使用了 $200 - |T_{\text{training}}|$ 项蜕变测试。最佳结果以**粗体** 显示。

		泛化率 G	
问题	$\|T_{\text{training}}\|$	**标准遗传编程**	**MTGP**
计数奇偶	25	**65.116**	62.5
	50	77.966	**81.395**
	100	91.429	**96.721**
	200	97.59	-
等级	25	4.819	**18.919**
	50	13.953	**19.737**
	100	28.395	**39.286**
	200	48.387	-
小或大	25	2.128	**12.069**
	50	13.924	**32.468**
	100	**38.462**	33.721
	200	48.315	-

率 G 为 18.919；对于小或大问题，MTGP 的泛化率为 12.069，而标准遗传编程分别仅为 4.819 和 2.128。

到目前为止，我们只研究了使用缩减后的带标签训练集的 MTGP。但即使运行过程中使用完整的带标签训练集（$|T_{\text{training}}| = 200$），泛化率是否仍能高于标准遗传编程？为回答这个问题，这次我们使用 800 个蜕变测试运行 MTGP。

表3：标准遗传编程和 MTGP 在所有所考虑的基准问题上达到的泛化率 G 。所有配置的带标签训练集大小均为 $|T_{\text{training}}| = 200$。对于 MTGP，除所考虑的带标签训练样例外，我们还使用了 800 个蜕变测试。最佳结果以**粗体** 字体显示。

		泛化率 G	
问题	$\|T_{\text{training}}\|$	**标准遗传编程**	**MTGP**
计数奇偶	200	97.59	**100.0**
等级	200	48.387	**61.538**
小或大	200	48.315	**62.069**

表3显示了所有受考察基准问题中，标准遗传编程和 MTGP 所达到的泛化率 G 。与之前一样，最佳泛化率 G 以**粗体** 显示。

我们看到，即使在一次运行中使用完整的带标签训练集 T_{training} ，采用蜕变测试也能提高泛化率 G。在所有研究的基准问题中，MTGP 的表现都最好。对于奇偶计数问题，我们甚至实现了 100% 的完美泛化率 G 。

总之，采用蜕变测试可以提高生成的程序的泛化能力。平均而言，MTGP 达到的泛化率最高。蜕变测试的主要优势在于，它们不需要耗费大量成本去手动计算预期输出，因为它们只使用可以自动生成的随机输入。

5 结论

遗传编程 [7,21] 是一种进化方法，以其在自动程序合成方面的出色表现而闻名。尽管遗传编程的性能可与其他最先进的程序合成方法相媲美 [26],，但它尚未成熟到足以应用于现实世界的软件开发，因为在训练过程中通常需要大量输入/输出示例，才能生成能够泛化到未见测试用例的程序。由于在实践中，训练用例必须由用户手动标注，成本很高，因此我们需要一种补充方法，用较少的人工定义训练用例来检查程序行为。

采用蜕变测试 [6], 时，我们不需要带标签的输入/输出示例。程序会执行多次：首先针对给定输入（可随机生成）运行，随后针对相关输入运行，以检查观测输出之间是否满足某些用户定义的蜕变关系。

因此，在本研究中，我们提出了 MTGP，这是一种结合蜕变测试与遗传编程的方法，并研究了它在常见程序合成基准问题上的表现以及生成的程序的泛化能力。此外，我们分析了泛化能力如何随给定训练用例数量而变化，并针对不同的带标签训练集大小进行了实验。

我们发现，在几乎所有实验配置中，将蜕变测试与人工标注的训练用例结合使用，都比单独使用人工标注的训练用例带来更高的泛化率；这些配置也包括基于遗传编程的程序合成中常见的较小带标签训练集。即使使用所考察的最大带标签训练集，在所有研究的基准问题上，采用蜕变测试也能大幅提高泛化率。因此，如果在所考虑的应用领域中，训练数据的标注成本高昂，我们建议研究人员在遗传编程方法中采用蜕变测试。

未来的工作中，我们将研究 MTGP 在更多程序合成基准问题上的应用，并进一步分析蜕变测试作为

运行期间提供的带标签训练用例的用途，以深入了解在遗传编程中引入蜕变测试的影响。

参考文献

1. Aenugu, S., Spector, L.：学习分类器系统中的词例选择。载于：遗传与进化计算会议论文集。第356–364页（2019） 2. Arrieta, A.：深度学习系统的多目标蜕变后续测试用例选择。载于：遗传与进化计算会议论文集。第1327–1335页（2022） 3. Błądek, I., Krawiec, K., Swan, J.：反例驱动遗传编程：基于形式化规范的启发式程序合成。《进化计算》**26**(3)，441–469（2018） 4. Chen, T.Y., Kuo, F.C., Liu, H., Poon, P.L., Towey, D., Tse, T., Zhou, Z.Q.：蜕变测试：挑战与机遇综述。《ACM计算综述》（CSUR）**51**(1)，1–27（2018） 5. Chen, T.Y., Kuo, F.C., Liu, Y., Tang, A.：蜕变测试与特殊值测试。载于：SNPD。第128–134页（2004） 6. Chen, T., Cheung, S., Yiu：蜕变测试：一种生成后续测试用例的新方法。香港科技大学计算机科学系，技术报告（1998） 7. Cramer, N.L.：简单顺序程序自适应生成的一种表示方法。载于：遗传算法与应用国际会议论文集。第183–187页（1985） 8. Fagan, D., Fenton, M., O'Neill, M.：探索文法演化中的位置无关初始化。载于：2016年IEEE进化计算大会（CEC）。第5060–5067页。IEEE（2016） 9. Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., O'Neill, M.：PonyGE2：Python中的文法演化。载于：遗传与进化计算会议论文集。第1194–1201页（2017） 10. Forstenlechner, S., Fagan, D., Nicolau, M., O'Neill, M.：适用于遗传编程中任意程序合成问题的语法设计模式。载于：欧洲遗传编程会议。第262–277页。Springer（2017） 11. Forstenlechner, S., Nicolau, M., Fagan, D., O'Neill, M.：基于推导树的遗传编程系统的语法设计。载于：欧洲遗传编程会议。第199–214页。Springer（2016） 12. Helmuth, T., Abdelhady, A.：对遗传编程程序合成中的亲本选择进行基准测试。载于：2020年遗传与进化计算会议论文集。第237–238页（2020） 13. Helmuth, T., Kelly, P.：PSB2：第二版程序合成基准套件。载于：遗传与进化计算会议论文集。第785–794页（2021） 14. Helmuth, T., McPhee, N.F., Pantridge, E., Spector, L.：通过自动简化提升演化程序的泛化能力。载于：遗传与进化计算会议论文集。第937–944页（2017） 15. Helmuth, T., McPhee, N.F., Spector, L.：用于程序合成的词例选择：多样性分析。载于：《遗传编程理论与实践 XIII》，第151–167页。Springer（2016）

16. Helmuth, T., McPhee, N.F., Spector, L.：使用统一的添加与删除变异进行程序合成。载于：遗传与进化计算会议论文集。第1127–1134页（2018） 17. Helmuth, T., Pantridge, E., Spector, L.：专家对于词例选择的重要性。《遗传编程与可进化机器》**21**(3)，349–373（2020） 18. Helmuth, T., Spector, L.：通用程序合成基准测试套件。载于：2015年遗传与进化计算年度大会论文集。第1039–1046页（2015） 19. Helmuth, T., Spector, L.：解释并利用下采样 lexicase 选择的优势。载于：ALIFE 2020：2020年人工生命大会。第341–349页。麻省理工学院出版社（2020） 20. Hemberg, E., Kelly, J., O'Reilly, U.M.：利用领域知识与新颖性，通过文法演化提升程序合成表现。载于：遗传与进化计算会议论文集。第1039–1046页（2019） 21. Koza, J. R.：遗传编程：通过自然选择进行计算机编程，第1卷。麻省理工学院出版社（1992） 22. Krawiec, K.：基于遗传编程的行为程序合成，第618卷。Springer（2016） 23. Langdon, W.B., Krauss, O.：通过软件数据维护将 sqrt 演化为 1/x。载于：2020年遗传与进化计算会议论文集。第1928–1936页（2020） 24. Schweim, D., Sobania, D., Rothlauf, F.：训练集大小的影响：程序合成中标准词例选择与下采样 lexicase 选择的比较。载于：2022年IEEE进化计算大会（CEC）。第1–8页。IEEE（2022） 25. Sobania, D.：文法引导遗传编程合成程序的泛化能力。载于：欧洲遗传编程会议（EvoStar的一部分）。第130–145页。Springer（2021） 26. Sobania, D., Briesch, M., Rothlauf, F.：选择你的编程助手：比较 GitHub Copilot 与遗传编程的程序合成表现。载于：遗传与进化计算会议论文集。第1019–1027页（2022） 27. Sobania, D., Rothlauf, F.：文法演化程序合成面临的挑战。载于：欧洲遗传编程会议（EvoStar的一部分）。第211–227页。Springer（2020） 28. Sobania, D., Rothlauf, F.：遗传编程程序合成的泛化能力度量。载于：遗传与进化计算会议论文集。第822–829页（2021） 29. Sobania, D., Rothlauf, F.：遗传编程程序合成：批次大小的影响。载于：欧洲遗传编程会议（EvoStar的一部分）。第118–129页。Springer（2022） 30. Sobania, D., Schweim, D., Rothlauf, F.：基于进化算法的程序合成综合调查。《IEEE进化计算汇刊》（2022） 31. Spector, L.：遗传编程中 lexicase 选择差异化表现对问题模态的评估：初步报告。载于：第14届遗传与进化计算年度会议论文集。第401–408页（2012） 32. Spector, L., Klein, J., Keijzer, M.：Push3执行栈与控制的进化。载于：第7届遗传与进化计算年度会议论文集。第1689–1696页（2005）

33. Spector, L., Robinson, A.：使用 Push 编程语言的遗传编程与自构造进化。《遗传编程与可进化机器》**3**(1)，7–40（2002） 34. Whigham, P.A. 等：基于语法的遗传编程。载于：遗传编程研讨会论文集：从理论到实际应用。第 16 卷，第 33–41 页。Citeseer（1995）