arXiv:2303.07166v1 [cs.LG] 13 Mar 2023

用于自动程序合成的改进树搜索

Aran Carmon [1] Lior Wolf [1]

摘要

在自动程序合成任务中，给定若干匹配的输入和输出对，需要生成一个特定领域特定语言（DSL）中的计算机程序，使其能够针对每个样本输入返回相匹配的输出。关键在于能够在有效程序的空间中进行高效搜索。本文提出一种蒙特卡洛树搜索（MCTS）的变体，在两种截然不同的领域特定语言上取得了最先进的结果。我们提出的探索方法包含多项创新：修改访问计数、对训练数据集进行预处理，以及对程序中已执行的部分进行编码。

1. 引言

搜索是许多机器学习任务的关键部分，这些任务的输出是一个序列。我们研究的是自动程序合成这一特定任务，其规范以输入/输出对的形式给出（Balog 等人，2017年；Kalyan 等人，2018；Gulwani，2016年；Singh 与 Gulwani，2015）。在这一序列生成任务中，与机器翻译和图像描述等其他生成任务类似，正确答案可能不止一个。

与自然语言处理中的大多数句子生成任务不同，在这项任务中，我们可以直接运行生成的程序来评估输出是否正确。将其视为强化学习问题时，这带来了明确且自然的奖励（Bunel 等人，2018）：生成的程序要么根据相应的输入产生指定的输出，要么不能。其他形式的奖励也可以考虑程序长度等因素，从而鼓励生成效率更高的程序。

[1]特拉维夫大学计算机科学学院。通信请联系：Lior Wolf <liorwolf@gmail.com>、Aran Carmon< arancarmon@mail.tau.ac.il>。

《机器学习国际会议强化学习探索研讨会论文集》，长滩，加利福尼亚州，PMLR 97，2019年。版权所有 © 2019，归作者所有。

然而，正如先前的研究所报告的（Zohar& Wolf，2018；Chen 等人，2019年），与使用最大似然损失进行训练、以生成作为真实标注的唯一程序相比，采用强化学习方法要么会损害性能，要么只能带来小幅提升。尽管最大似然估计方法采用教师强制的方式训练，但情况依然如此：在训练期间（与测试时不同），所考虑的部分程序是真实程序的前缀。

因此，在这项工作中，我们聚焦于最大似然估计方法，并考察不同的树搜索策略。这些方法包括经典的束搜索，它常用于程序合成领域；CAB 变体，它过去曾被成功使用（Zohar& Wolf，2018）；以及我们开发并探索的各种蒙特卡洛树搜索方法。

除了研究将蒙特卡洛树搜索应用于这一问题时所涉及的具体构建模块外，我们还提出了另外两项改进。第一项是对训练集进行预处理，第二项是添加一个用于编码当前程序历史的编码器。我们展示了这些技术如何分别使用或结合使用，从而提高所得准确率。

2. 相关工作

我们在两种领域特定语言（DSL）上进行了实验。第一种是 Deep-Coder DSL（Balog 等人，2017年；Zohar& Wolf，2018），其中会对整数寄存器进行操作。每个寄存器可以存放一个整数或一个整数列表，程序可以对寄存器应用排序、取尾部和乘法等函数。 Balog 等人 (2017年) 利用神经网络对输入/输出对进行预测，以增强SMT求解器和枚举搜索等经典程序搜索技术。具体而言，神经网络会生成一个概率向量，表示程序中每条命令或每个函数存在的概率。后来，Zohar& Wolf (2018) 使用神经网络预测在应用部分程序后、束搜索过程中，给定执行状态时下一条语句的概率。具体而言，他们采用了CAB搜索策略（Zhang，1998年）。由于在这种领域特定语言中，每条语句都会向内存中返回一个整数或列表，而内存容量有限，因此采用了学习得到的垃圾回收

机制来从内存中丢弃先前语句的结果。我们针对这种领域特定语言开展的工作扩展了（Zohar& Wolf，2018）的方法，并且也采用了这种垃圾回收器。

Karel 领域特定语言（Devlin等人，2017年；Bunel 等人，2018；Chen 等人，2019年）作用于一个包含执行体、各种标记物和障碍物的二维网格世界。Karel 领域特定语言包含条件语句和循环语句，而这些并非 DeepCoder 领域特定语言的一部分。不过，后者包含一些额外的高级函数，例如排序或映射。Bunel 等人（2018）展示了如何通过强化学习改进该领域特定语言的结果。Chen 等人（2019年）提出了当前执行状态的编码方式（与Zohar & Wolf （2018）类似），并采用了集成方法。在我们的工作中，我们不研究集成方法的影响。

束搜索或许是探索搜索树最流行的方法。它本质上类似于最佳优先搜索，但最佳优先搜索只将最有希望的候选项保留在内存中，而束搜索会同时保留多条候选轨迹。候选项的数量称为束宽；从每条候选轨迹中提取的派生扩展数量（在剪枝至束宽之前）称为提取宽度。

完全随时束搜索（CAB）Zhang，1998年）是束搜索的一种扩展，更适合在固定时间预算下进行搜索。在束搜索中，如果束宽和扩展规模设置得过低，搜索就会过于薄弱，无法到达目标；如果设置得过高，则会在实现目标之前达到执行时间上限。CAB通过重复运行束搜索来扩展它：从较弱且成本较低的设置开始，并在每次搜索失败后提高搜索参数，只要仍有时间运行，就继续如此。

蒙特卡洛树搜索（MCTS）Chaslot等人，2007）是一种广受欢迎的方法，用于在游戏搜索树中优先进行探索。搜索会逐步扩展，兼顾有前景节点和未探索节点。它包括以下步骤：(1) 选择：遍历搜索树，直到到达一个新的叶节点。(2) 扩展：将该叶节点添加到树中。(3) 模拟：评估叶节点的价值。(4) 反向传播：根据叶节点的价值，更新通往该叶节点路径上的父节点的值。在我们的工作中，我们不使用价值函数，也不采用最后两个步骤。

人工智能领域的一项重大里程碑，是通过深度学习、强化学习和蒙特卡洛树搜索（Silver等人，2016年；2017年）实现了围棋的超人类表现。我们的方法与之共有的具体部分，是基于访问计数。不过，我们主张在环境层面进行计数，而以往围棋研究中的计数则基于树中的位置，而非棋盘布局。

基于树中的位置，而不是棋盘布局。

我们的方法不使用学习得到的价值函数，也不使用基于模拟（展开）的价值函数，这与Silver等人(2017年)的做法不同。我们曾尝试学习此类价值函数，但发现结果并未改善。这可能是因为最终目标，即程序的输出各不相同，使得这类学习过于困难。此外，正如在使用强化学习解决魔方问题时所指出的（McAleer等人，2018），价值函数主要缩减蒙特卡洛树搜索的深度，而非广度。在我们的情形中，由于程序长度有限，深度并不那么重要。

3. 方法

我们的方法基于使用蒙特卡洛树搜索，而非束搜索或CAB；后两者是先前工作的搜索技术。我们采用了与（Zohar& Wolf，2018）相同的网络。编码器 E 会对执行过程的输入状态以及每个样本的输出进行嵌入，并使用平均池化来获得单个状态向量。基于该向量，网络 P 预测下一条语句，网络 G 则预测要丢弃的变量。P 和 G 共享大部分层。在我们的工作中，我们提出将执行至当前状态的部分程序的嵌入作为额外输入加入网络 P，见第3.3节。

3.1. 我们的蒙特卡洛树搜索变体

给定一个训练好的网络 P，它能够优先选择最有可能的下一条命令，我们便可以通过搜索树探索程序空间。树中的节点对应于选择语句后的执行状态。搜索会优先考虑过去探索较少的节点（探索），以及由预测为可能性较高的命令所产生的节点（利用）。使用分数 $U(s,a) = {}^{P}(s,\ a)\ N(s,\ a)+1$，其中 $P(s,a)$ 是训练好的网络的输出（在给定状态 s 时选择动作 a，即命令的伪概率），而 $N(s,a)$ 表示在状态 s 下采取动作 a 作为响应的次数。

与我们所了解的文献不同，在我们采用的搜索变体中，我们不会回溯。相反，我们每次都从根节点开始搜索。首先，我们根据网络的输出选择最有希望的命令，直到找到解、遇到无效的执行状态，或达到程序的最大长度限制。如果未找到解，我们就重新开始搜索，这次使用更新后的分数U，以降低已访问节点的权重。

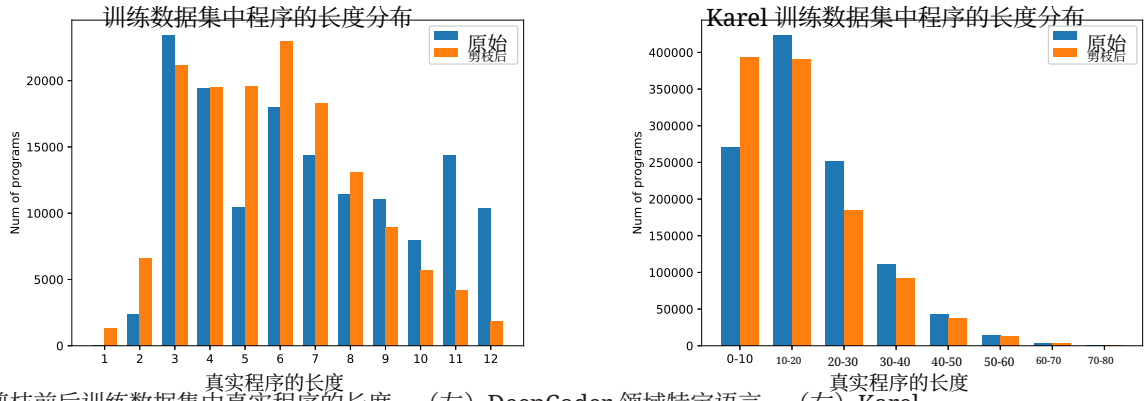


图 1. 剪枝前后训练数据集中真实程序的长度。（左）DeepCoder 领域特定语言。（右）Karel。

共享计数 与我们所了解的蒙特卡洛树搜索方法按树节点统计访问次数不同，我们发现统计每个状态被访问的次数很有用。

状态是指在输入/输出规范中的每个输入上运行部分程序（从根节点到当前节点）后得到的内存状态。如果规范中的所有输入所对应的内存状态都相同，则这两个状态相同。

在 DeepCoder 领域特定语言中，内存包含各种寄存器。我们对寄存器进行排序，使得即使寄存器顺序有所不同，只要底层值相同，两个状态就相同。在 Karel领域特定语言中，内存包含网格世界的状态。

3.2. 剪枝

由于多种执行路径会产生相同的状态，使用最大似然估计损失进行学习可能并非最优。先前的研究（Bunel 等人，2018）建议采用强化学习来解决这一问题；然而，正如后续研究所指出的，这并不一定能显著提升性能。

相反，我们建议预处理训练集，将程序替换为更短且符合输入/输出规范的程序。为完成这一任务，我们使用一个基于原始训练数据预训练的网络。使用我们改进的蒙特卡洛树搜索变体将该网络应用于训练数据集时，如果找到一个更短且符合规范的程序，我们就用它替换真实标注程序。完成这一步剪枝后，再从头开始训练。

剪枝会显著影响训练数据集中的程序长度，如图 1 所示。例如，对于 DeepCoder 领域特定语言，它将长度为 12 的程序缩短了 82% ；而长度为 12 是训练数据集中可用的最大长度。

3.3. 部分程序的编码

PCCoder（Zohar & Wolf，2018）中，网络P的输入是执行环境当前寄存器的嵌入（每个样本输入/输出对应一个环境），其中包括每个样本的目标（输出）寄存器；该网络用于预测下一条语句。对于 Karel，我们采用了与 PCCoder 类似的架构。在这种情况下，P 接收输入/输出网格的嵌入，以及最后执行的命令的嵌入；后者是根据Bunel 等人（2018）的工作添加的。

在这项工作中，我们提出添加一个额外的输入，用于编码搜索过程中已构建的部分程序此前执行的命令。

对于 DeepCoder 领域特定语言，部分程序表示为由四元组构成的序列，每个四元组包含一个运算符索引、参数索引（一个或两个；如果运算符只接受一个参数，则将其复制）和一个输出索引。我们对这四个索引应用嵌入层。

随后，将长短期记忆网络应用于四个嵌入的拼接结果。隐藏单元的数量设为我们用于索引嵌入的各个查找表中选项数量之和（66）。然后，将序列末尾的隐藏状态与来自 P 其他输入的激活值拼接；这些输入已通过 P 的 5 个 DenseNet 层。接着应用两个大小相同的全连接层，之后再应用一个维度与原始 P 相同的 Softmax 层。

在 Karel 中，输入和输出网格采用与（Bunel 等人，2018）所用的相同卷积网络进行编码。部分程序通过运行两个长短期记忆网络来编码：第一个处理命令的嵌入（命令没有参数），第二个处理嵌套层级（即命令是否在 if 条件、while 循环等结构内执行）。查找表包含 38 种命令选项和六种嵌套选项——

。长短期记忆网络的隐藏单元数量与相应选项数量一致。序列末尾两个长短期记忆网络的隐藏状态与嵌入后的输入/输出样本（经过卷积神经网络编码后）拼接，并通过三个线性层处理，每层有 512 个隐藏单元。该过程针对每个输入/输出样本分别进行，随后通过最大池化聚合，最后经过 Softmax 层。

4. 实验

我们测试了各种搜索方法，并评估了各项改进对性能的影响。在所有实验中，我们均采用 PCCoder 架构。将此方法应用于 Karel 时，使用了与（Bunel 等人，2018）相同的输入/输出编码。与先前的工作不同，我们不使用长短期记忆网络来生成程序（PCCoder 只预测下一个命令），因此以全连接网络取代了长短期记忆网络。该领域特定语言中不需要垃圾回收网络 G。

请注意，只要时间充足，即使采用朴素的搜索方法，也能找到与输入/输出对一致、长度不超过某个上限的程序。随后可以选取其中最短的程序，以提升泛化能力。因此，必须设定允许的最长运行时间。这在比较各项改进时显然存在弊端。然而，在使用相同标准硬件的情况下，固定的运行时间预算能够提供可靠的性能指标。在我们的实验中，所有实验均在商业云基础设施上进行。

DeepCoder 领域特定语言实验采用了八个寄存器，并使用了（Zohar& Wolf，2018）的基准，其中包含长度最多为 12 的训练程序，以及长度为 14 的真实测试程序（所有测试环境均由长度为 14 的程序构造，这些程序可能有也可能没有更短的等价程序）。

结果见表 1。如图所示，我们的所有改进（对数据集进行剪枝、对部分程序进行编码、用我们改进的蒙特卡洛树搜索变体替换 CAB，以及共享计数蒙特卡洛树搜索）都提升了相较于基线的性能。这一结论在三种不同的超时设置下均成立。我们包含所有改进的最佳结果准确率为 85.2%，而先前工作的准确率为 50.2%；两者的超时设置相同，均为 2000 秒。仅部分程序编码一项就使 CAB 搜索的准确率提高了 15%。剪枝的效果则不那么显著：它使 CAB 的准确率提高了 3%。

此外，剪枝本身并不会增加蒙特卡洛树搜索方法中的共享访问计数。不过，与部分历史记录编码结合后，它会带来显著提升，例如在 500 秒超时设置下提升 5.2%。

在通过是否应用剪枝或部分程序编码得到的四种组合中，蒙特卡洛树搜索方法

表1.DeepCoder 基准测试上的准确率（百分比）。pruned 列表示训练集是否经过预剪枝。PartEnc 表示是否将部分程序的编码添加到预测网络中。 P比较了三种搜索策略（束搜索的表现不具竞争力），其中 shared 表示采用共享计数的蒙特卡洛树搜索。测试了不同的超时值。 [1]这些结果与（Zohar & Wolf，2018）的结果相对应。

			超时		
剪枝后	部分历史编码	搜索	500	1000	2000
No	No	CAB[1]	39.4	44.2	50.2
		MCTS	59.4	65.2	69.6
		共享	67.0	72.4	76.0
No	Yes	CAB	54.4	59.6	65.2
		MCTS	69.6	75.4	80.2
		共享	72.2	78.8	83.0
Yes	No	CAB	43.8	48.8	53.6
		MCTS	59.0	63.2	68.8
		共享	64.8	69.0	72.6
Yes	Yes	CAB	62.4	65.8	71.8
		MCTS	74.4	79.0	82.2
		共享	77.4	81.2	85.2

表 2. Karel 基准测试上的准确率（百分比）。蒙特卡洛树搜索+和共享+ 表示在搜索方法中同时加入数据集剪枝和部分程序编码。

	超时=1000		超时=2000	
方法	5个目标	包含留出集	5个目标	包含留出集
Beam	86.28	81.00	86.28	81.00
CAB	88.24	82.72	89.40	83.60
MCTS	87.24	81.76	88.96	83.08
蒙特卡洛树搜索+	89.20	83.04	90.08	83.68
共享	87.16	80.96	87.84	81.48
共享+	89.04	83.00	89.88	83.64

采用共享计数的方法以显著优势胜过其他搜索方法，而不采用共享计数的蒙特卡洛树搜索变体则优于 CAB。

所用的 Karel 基准测试取自（Devlin等人，2017年），该基准测试也被（Bunel 等人，2018；Chen 等人，2019年）使用。然而，在这一基准测试上，我们发现很难与以往工作直接比较，因为以往工作没有采用超时机制，而是使用固定数量的束，这与我们的蒙特卡洛树搜索变体每次搜索失败后都返回根节点的做法不兼容。

通过向搜索器提供五个输入/输出样本，并测量搜索器能否成功找到一个程序来评估性能。该程序不仅要与这五个输入/输出示例一致，还要能泛化到第六个未见过的输入/输出样本。在表2中，我们给出了这两个目标的准确率（分别是与五个或六个样本的一致性）。结果分别对应 1000 秒和 2000 秒的超时设置

。束搜索的结果采用我们找到的最佳束宽和扩展参数（分别为 128 和 5）。CAB 参数也经过优化，以最大限度地提高测试准确率；它从类似的束搜索开始，在每次重复尝试时将束宽加倍，并将扩展规模增加 1。

对于这种领域特定语言，结合使用蒙特卡洛树搜索、过往编码和数据集剪枝，在两种超时设置和两项成功标准下均取得了最佳结果。在相同世界网格之间共享访问计数并未改善结果，而新增的簿记操作增加了运行时间，导致结果略差。

致谢

本研究获得了一项 ICRC 资助。

参考文献

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., 和 Tarlow, D. Deepcoder：学习编写程序。载于*ICLR*，2017年。Bunel, R., Hausknecht, M., Devlin, J., Singh, R., 和 Kohli, P. 利用语法和强化学习进行神经程序合成。arXiv 预印本 arXiv:1805.04276，2018。Chaslot, G., Winands, M., Uiterwijk, J., Van Den Herik, H., 和 Bouzy, B. 蒙特卡洛树搜索的渐进式策略。载于第十届联合信息科学会议论文集（JCIS 2007），第 655–661 页，2007年。Chen, X., Liu, C., 和 Song, D. 执行引导的神经程序合成。载于*ICLR*，2019年。Devlin, J., Bunel, R. R., Singh, R., Hausknecht, M., 和 Kohli, P. 神经程序元归纳。载于神经信息处理系统进展，第 2080–2088 页，2017年。Gulwani, S. 示例编程：应用、算法与歧义消解。载于第八届国际自动推理联合会议论文集——第 9706 卷，第 9–14 页，柏林、海德堡，2016年。施普林格出版社。ISBN 978-3-319-40228-4。Kalyan, A., Mohta, A., Polozov, O., Batra, D., Jain, P., 和 Gulwani, S. 神经引导的演绎搜索：从示例中实时合成程序。载于*ICLR*，2018。网址：https://openreview.net/forum?id=rywDjg

–RW.

McAleer, S., Agostinelli, F., Shmakov, A., and Baldi, P. 无需人类知识解决魔方。arXiv 预印本 arXiv:1805.07470, 2018。

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. 使用深度神经网络和树搜索掌握围棋。《自然》，529(7587):484，2016年。Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. 无需人类知识掌握围棋。《自然》，550(7676):354，2017年。Singh, R. and Gulwani, S. 在示例编程中预测正确的程序。载于 CAV，2015年7月。Zhang, W. 完全任意时刻束搜索。载于 AAAI，第 425–430 页，1998年。Zohar, A. and Wolf, L. 使用习得的垃圾回收器自动合成长程序。载于《神经信息处理系统进展》，第 2094–2103 页，2018。

Deepcoder

Input: [1, -7, -14, 12, 8, 18, -11]

Output: [-14, -22, -28]

```
Filter(<0)
Map(*2)
Sort()
Reverse()
```

RobustFill

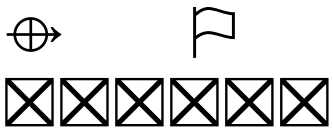
Input	Output
January	J-jan
February	F-feb
March	M-mar

```
Program = Concat(e1, e2, e3)

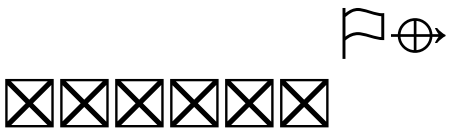
e1 = SubStr(0,1)
e2 = Const(-)
e3 = ToCase(Lower,SubStr(0,3))
```

Karel

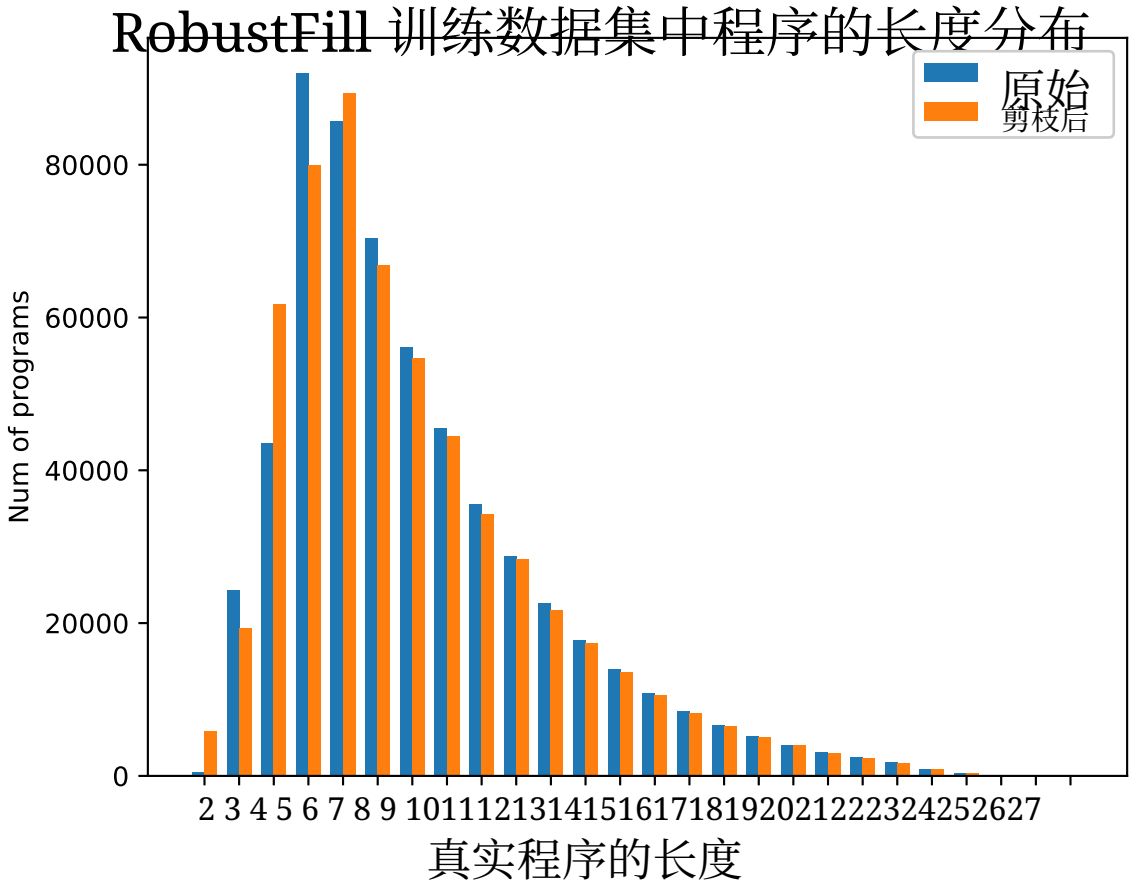
Input

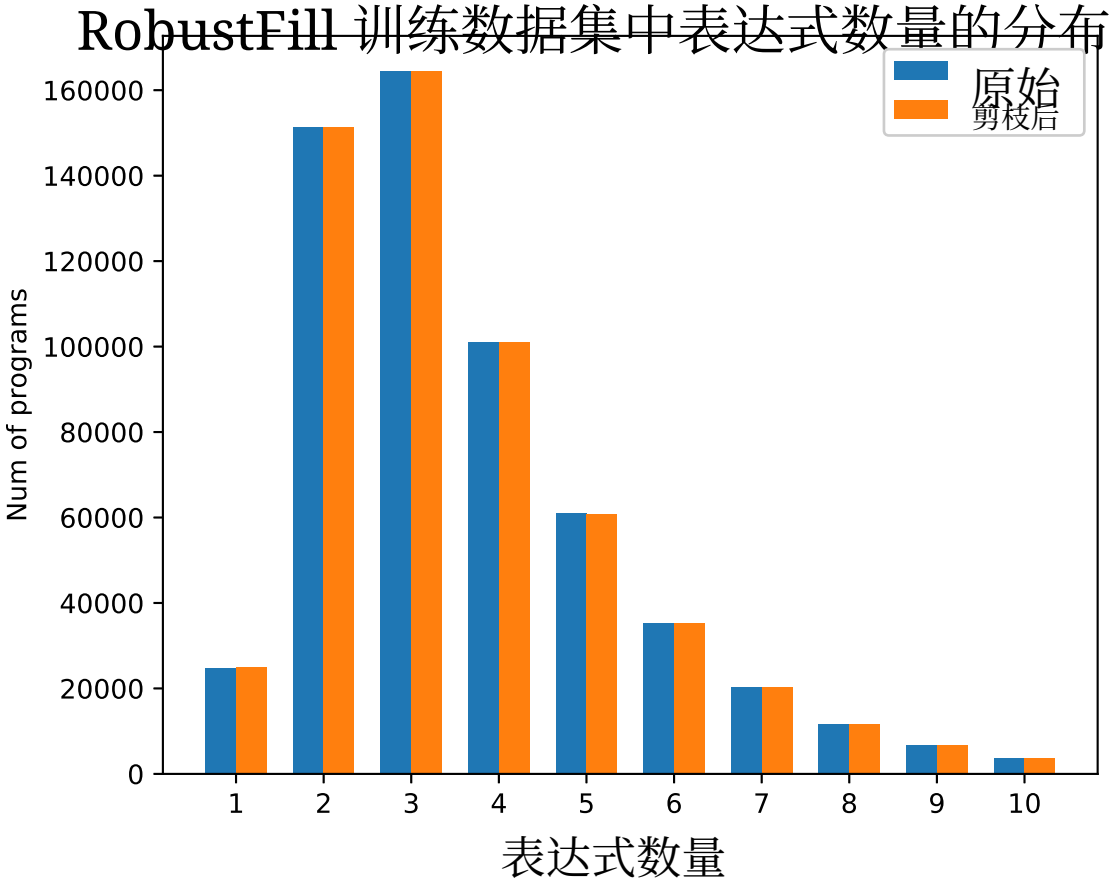


Output



```
While (rightIsNotClear) {
     If (markerPresent) {
          pickMarker()
     }
     move()
}
putMarker()
move()
```


RobustFill 训练数据集中程序的长度分布
原始
剪枝后
Num of programs
0
20000
40000
60000
80000
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
真实程序的长度


RobustFill 训练数据集中表达式数量的分布
原始
剪枝后
Num of programs
0
20000
40000
60000
80000
100000
120000
140000
160000
1
2
3
4
5
6
7
8
9
10
表达式数量