1. 使用MAP-ELITES探索基因编程

1.1引言

当程序员编写代码时，理想情况下，他们希望程序能够快速实现、易于扩展和清晰地供他人理解。当然，这些属性通常是不兼容的:编写速度最快的程序架构通常不容易扩展或理解。当我们用遗传程序(GP)进化程序时，我们面临着类似的问题;最容易进化的解决方案通常是一团乱麻。它们不能作为解决更复杂问题的构建模块(也就是说，它们是不可进化的)，而且它们对梳理出正在发生的事情具有挑战性。因为程序体系结构对于可演化性和可分解性是如此重要，所以大量的工作都投入到开发遗传编程系统中，以促进具有更可演化的体系结构的程序的演化。例如，模块化是软件设计的重要原则，也是可演化性的重要组成部分，这导致许多人以促进模块化代码进化为目标设计遗传编程系统。

事实上，在基因编程中，我们有大量的方法来表示我们的程序，每一种都有其独特的一组可用的程序元素，以及组织、解释和改变程序的方法。考虑到GP表示的多样性，了解如何为特定问题选择最合适的表示或表示配置是目前领域的一个仍待研究的问题。在这个问题上取得进展需要为GP表示扩展现有的形式分析工具包。虽然可以考虑代码的许多不同的高级属性，但在本章的其余部分，我们将关注代码的可演化性。

特别是，有一种方法来深入了解一个给定的表示能够演进的程序体系结构的范围是很有帮助的。这样做将有助于我们将与表示本身相关的问题与进化算法其余部分的设置方式的问题分开(众所周知，促进可进化性的选择具有挑战性)。此外，访问具有不同架构的程序示例对于建立实验至关重要，这些实验将告诉我们这些架构何时有用，以及它们如何与给定表示的其他特性交互。最终，一个用于探索程序架构的工具将帮助我们得出可能对该领域的其他人有用的一般化见解。

想要访问具有一系列不同体系结构的程序的问题是一个常见问题的实例。通常在进化计算中，我们希望进化出不同于一系列表型性状[18]的好解决方案。例如，我们可能想要为做出决策的涉众提供各种选项[4]。另外，我们可能想要为机器人提供替代的运动策略，以便在它受损时使用[5]。MAP—ELITES已经被证明是一种有效的技术，可以为一个问题[15]演化出一套不同的解决方案。

在MAP-ELITES中，用户选择一些他们期望与解决问题相关但可能与适合度不直接相关的表型轴。然后，每个轴被离散到预先确定数量的箱中，形成一个多维网格，网格中的一个位置对应着表型性状的独特组合。当产生一个新的溶液时，评估它的表型，并将其放入与该表型相对应的容器中。如果这个箱子是空的或者被一个低健康水平的人占用了，新的解决方案就会取代它。否则将被丢弃。

到目前为止，MAP—ELITES已经被用于进化机器人手臂[6]，机器人步态[5]，软体机器人[15]，以及用于计算机视觉任务[15]的神经网络。有趣的是，在最后一个任务中，表型轴(连接成本和模块性)与神经网络本身的形态有关。使用这些坐标轴，MAP—ELITES不仅找到了一系列好的解决方案，而且提供了与这两个特征相关的潜在健身景观的拓扑结构的洞察力。由MAP—ELITES制作的热图显示了哪些类型的网络能够成功完成任务，以及网络特征对彼此的限制。在这里，我们试图对全科医生做同样的事情。解决一个问题是否有多种编程途径?我们能否识别在演进程序中不同特性(如模块性、指令组成等)之间的内在权衡?我们将MAP—ELITES视为一种工具，它可以通过阐释GP表示的不同方面在应用于问题时的交互来回答这些问题。这种加深的理解有助于建立一种直觉，了解什么类型的程序可能最适合给定的问题，这可以用于通知表示选择、填充初始化或变异操作符。

在本章中，我们将演示如何使用地图精英来探索一个简单的线性GP表示。通过为对应于程序架构和指令组成的地图精英选择表型轴，我们可以展示GP表示的不同特征与跨越各种问题的进化过程的相关性。我们比较了地图精英、词典选择、比赛选择和随机漂移下的程序形式，表明地图精英产生了更多样的解决方案。此外，我们还讨论了其他可能在地图精英中用作表型轴的程序性状，但我们还没有在这项工作中进行探索。

1.2 MAP—ELITES方法

1.2.1基本组成

在这项研究中，我们进化了线性遗传程序，其中每个程序都是指令的线性序列，每个指令最多有三个参数，可以修改其行为。最值得注意的是，我们的线性遗传规划(LGP)计算基板支持子程序，允许具有模块化架构的程序进化。

长期以来，人们一直认为有效利用模块化子程序对促进解决复杂问题的遗传程序的进化非常重要。的确，将模块整合到GP中已经被广泛地探索过，它们的好处也被很好地记录了下来(例如，[1,8,10,11,20,21,24,26])。我们设计LGP表示是为了促进模块化和可重用代码的进化，同时最小化影响传统线性遗传程序的组织方式(即，线性指令序列)。我们在语言中包含了许多自动创建编程结构(如循环和子例程)的指令，并且我们通过“作用域”的概念启用了这些功能，这在功能上与Push中的“环境”类似。

1.2.1.1虚拟CPU硬件

我们的线性遗传程序是在虚拟CPU的环境中执行的，包含以下组件:

•指令指针:指示当前正在执行的基因组位置的标记。许多指令将影响指令指针(IP)在基因组中移动的方式。

•寄存器:每个虚拟CPU有16个寄存器。程序可以在每个寄存器中存储单个浮点值。寄存器用与其ID对应的数字进行初始化(例如，寄存器0初始化为0.0，寄存器1初始化为1.0，等等)。

•栈:每个虚拟CPU有16个栈。程序可以把浮点数推到这些堆栈上，然后再把它们取出来。

•输入:每个虚拟CPU可以接受任意数量的输入值。这些值不需要按任何特定的顺序排列，但是每个值都需要与程序可以用来访问它的唯一数字标签相关联。出于本文的目的，我们总是使用从0开始的顺序整数。

•输出:输出与输入功能相同。输入和输出之间唯一的区别是指令与它们交互的方式;指令可以从输入端读，但不能从输入端写，指令可以写入输出端，但不能从输出端读。

•作用域:每个虚拟CPU有16个作用域(加上全局作用域)，将在下一节中描述。

1.2.1.2范围

在软件开发中，变量的范围指定了该变量可以使用的代码区域。从某种意义上说，作用域就像一个编程膜，能够封装所有方式的编程元素，比如变量、函数等等，并允许整个区域循环或跳过。我们的LGP表示为不断发展的程序提供了对指令和内存作用域的控制，允许程序轻松地管理流控制和可变的生命周期。

在我们的LGP表示中，作用域提供了主干，所有其他模块化促进特性(如循环和函数)都是在其之上构建的。程序中的所有指令都存在于一个作用域中，可以是默认的最外层作用域，也可以是其他16个可用作用域中的一个(这使得17个可能的作用域可以通过各种指令访问)。这16个内部作用域都有一个层次结构，这样编号较高的作用域总是嵌套在编号较低的作用域中。

在程序开始时，在第一个作用域更改之前的所有指令都在最外层作用域中。在基因组中发生作用域更改指令之后，后续指令被添加到新的作用域，直到遇到另一条作用域更改指令，依此类推。这些作用域按数字顺序排列。编号较高的范围总是嵌套在编号较低的范围内。可以使用break指令退出作用域，也可以使用任何将控制移到编号较低的作用域的指令退出作用域。

作用域也是程序模块(函数)的基础。define指令允许程序将指令放入一个作用域，并将该作用域的内容与16个可能的函数名中的一个关联起来。之后，如果调用该函数(使用call指令)，程序将进入定义该函数的作用域，并依次执行该作用域内的指令，包括任何内部(嵌套)作用域。

类似地，作用域是循环的基础。在这里使用的指令集中存在两种循环:while循环和倒计时循环。这两种类型的循环都有相应的作用域，其中包含组成循环体的指令序列。这两种类型的循环都会重复它们的循环体(即相关作用域的内容)，直到参数指定寄存器中的值为0。倒计时循环在每次迭代时自动减少这个寄存器1。当遇到任何会导致程序离开当前作用域的指令时，就结束当前迭代，开始下一个迭代。

1.2.1.3指令

在这项工作中，演进程序可以包含以下26条不同指令的库。对于每个指令参数，都限制为16个值(0到15)，用于指定以下任意一个值:寄存器、作用域、函数、输入或输出。每个指令及其类型的参数都显示在指令名称后的括号中。

报纸上的文字

描述已自动生成

文本, 信件

描述已自动生成

1.2.2进化

1.2.2.1选择操作

**MAP-ELITES**

该算法旨在阐明搜索空间，并已被证明是一种有效的技术，用于进化一个问题[15]的多样化解决方案集。在地图精英中，一个种群是基于一组选择的表型性状构建的。每个选择的性状在网格的细胞上定义一个轴，每个细胞代表选择的性状的不同组合。细胞只保留最适合的(精英)解决方案，发现与该细胞相关的特征组合。通过随机生成解决方案并将它们放置到网格中相应的单元中(基于随机解决方案的特征)，可以初始化地图精英网格。初始化后，被占用的细胞被随机选择复制。当一个解决方案被选择用于繁殖时，我们会产生一个突变的后代，并确定该后代在网格中的位置。如果细胞是空的，新的溶液被放置在那个细胞中;否则，我们将新的解决方案与当前居住者的适应性进行比较，保持两者的适应性。随着时间的推移，这个过程产生了一个解决方案的网格，跨越了我们用来定义网格轴的特征范围。

**Tournament Selection**

为了理解使用MAP-ELITES来探索潜在程序架构的范围是否有价值(而不是简单地查看在已经被使用的选择方案下发展的架构)，我们需要将其与一个更标准的选择方法进行比较。在这里，锦标赛大小为2的锦标赛选择将作为该控制。当我们需要使用竞赛选择来生成后代程序时，我们从种群中随机选择两个程序，并允许更适合的程序进行繁殖。我们有意选择尽可能小的比赛规模，以减少选择的强度，尽可能促进多样化。

**Lexicase 选择**

众所周知，Tournament Selection在维持种群多样性方面表现不佳。我们知道，词汇酶选择[23]可以保持表型多样性，尽管我们对这些群体中程序体系结构的多样性知之甚少。为了更好地理解我们所提议的使用MAP-ELITES是否比简单地保持人口多样性更有效地探索可能的程序架构范围，我们将其与使用标准词典选择的结果进行比较。

在词汇选择中，一个问题的所有测试用例在每次选择另一个程序进行复制时被随机重新排序。然后，我们按顺序遍历测试用例，对于每个测试用例，从候选程序集中删除除性能最好的程序外的所有程序。当只剩下一个程序时，我们允许它复制。如果打成平手，我们就随机挑选。

**Random Drift**

在没有选择压力的情况下，会出现什么类型的程序架构呢？此外，我们还比较了地图精英生成的演进程序架构的范围和随机漂移生成的程序架构的范围，在随机漂移中，我们选择程序进行随机复制。尽管我们并不期望这些程序能够真正解决我们的测试问题，但它们将提供我们在没有选择的情况下所期望的大规模统计趋势的洞见。

1.2.2.2变异操作

在这项工作中，我们无性繁殖程序，并应用一致的突变率在所有的治疗后代。我们使用了四种不同的操作符来在复制时引入突变:指令替换、参数替换、点插入和点删除。指令替换，即一条指令被另一条指令随机替换，在基因组的每个位点发生的几率为0.005。参数替换，即指令的一个参数被另一个参数随机替换，在基因组中的每个参数发生的概率为0.005。在点插入中，一个随机的指令被添加到一个给定的位置后，增加了基因组的长度。相反，点删除删除了给定位置上的指令，减少了基因组的长度。点插入和点删除都有0.005的几率发生在基因组的每个位点上。

1.2.3实验设计

1.2.3.1表型轴

作为概念证明，我们选择了两个表型轴，我们期望促进对我们的线性GP表示的特征的有用的探索。在8.13节中，我们将讨论其他坐标轴，这些坐标轴也可能被证明对探索GP表示通常是有用的。

**项目组成**

表示的指令集对程序能够做什么有巨大的影响。然而，预先预测一个指令的重要性是具有挑战性的。地图精英可以帮助理解指令在不同情境中的重要性。从理论上讲，与指示相关的多种表型性状都可以使用。例如，对于任何一条指令，它的使用次数可以是一个轴。

然而，为了从高层次上理解可以进化的程序的范围，我们选择将程序中指令的总体多样性作为一个轴。在这里，我们使用香农熵量化程序中指令的多样性。这种测量提供了关于基因型整体的高级信息。重要的是，它不可能通过少量的中性突变轻易改变，这意味着它应该提供关于基因组之间实际差异的信息。我们将这个值离散到从0到程序的最大可能熵之间的20个bins中。

**模块使用**

我们的表示以作用域的形式围绕模块展开。当我们试图理解这种编程范式对进化是否有用时，理解作用域的使用程度是至关重要的。因此，我们选择程序使用的范围的数量作为我们的第二个表型轴。重要的是，我们只计算实际使用的作用域;当程序运行时，它必须在给定的作用域中执行至少一条指令才能获得使用该作用域的学分。然而，这并不能保证作用域的使用是有意义的，只能保证它们被使用了。因为这个度量已经是一个整数值，所以我们沿着这个轴使用了17个箱子，以便每个箱子对应不同可能数量的作用域。

1.2.3.2测试问题

除了逻辑问题之外，所有的问题都是由一组测试用例定义的，在这些测试用例中，程序被给予指定的输入数据，并根据它们的输出与正确输出的接近程度评分。对于地图精英和比赛选择，我们用这些测试案例的分数来计算适合度。

•逻辑:程序接收两个二进制形式的整数，必须输出对它们进行位逻辑运算的结果。我们奖励10个2输入(除了ECHO，它是1输入)逻辑操作:ECHO、NOT、NAND、OR-NOT、AND-NOT、NOR、XOR和EQUALS。为了促进这些计算的发展，我们在指令集中添加了一个Nand指令，它将输入转换为整数，然后执行一个按位的非与操作，结构与Add指令相同。在程序执行过程中，程序输出解的每一个唯一逻辑操作都使该程序的得分增加1。一旦一个程序解决了所有的逻辑问题，它就会因为解决这些问题的速度而获得额外的分数。具体来说，这个奖励等于允许执行的指令总数减去程序在执行所有10个逻辑任务之前实际执行的指令数。对于词汇选择，每个逻辑操作都被视为一个不同的测试用例。

•平方:程序接收一个整数作为输入，并且必须输出它的平方。因为这个问题很简单，我们只用11个测试用例来评估程序。

•Sum:程序接收一个包含5个整数的列表作为输入，它们必须相加并输出它们的总数。程序是根据一组200个测试用例进行评估的。

•最小:程序接收一个包含四个整数的列表作为输入，并且必须输出最小的一个。程序是根据一组200个测试用例进行评估的。

1.2.3.3实验参数

我们在每种条件下进行30个重复，共5万代。在使用竞赛选择、词汇选择或随机漂移来确定哪些程序重现的情况下，我们保持了1000个程序的种群规模。然而，地图精英中的最大人口规模取决于用于定义地图精英网格的表型性状轴的数量和分辨率。因此，在使用地图精英的情况下，最大人口规模是340人。然而，在我们的map -精英条件中，我们将单个代定义为等于1000个复制事件，这确保了所有条件经历相同数量的复制事件。

我们通过生成长度随机的随机程序来初始化总体。程序的长度不能小于1条指令，也不能大于1024条指令。每一个程序的评估都是通过按顺序执行其指令，直到达到上限(方框和逻辑问题执行128条指令;512条指令执行的总和和最小的问题)。

1.2.3.4数据分析

量化不同范围的程序架构探讨不同选择运营商我们看最后一代又一代的人口我们所有的复制和过滤掉所有的程序不能完全解决这个问题(例如,那些不完美在所有测试用例,或者上下文的逻辑问题,不执行所有的逻辑运算)。然而，在随机漂移条件下，我们不会从最终的种群中过滤任何程序，因为指望它们能够解决问题是不合理的。然后，我们查看每个表型轴的值分布，并使用Kolmogorov-Smirnov检验在选择方案中比较它们，以告诉我们地图精英是否产生了与其他选择方案显著不同的程序架构分布。为了纠正我们执行的Kolomogorov-Smirnov测试的数量(对于我们比较map -精英的每个备选选择操作符，一个测试，对于范围计数和指令熵)，我们使用Bonferonni校正。所有数据分析使用R统计计算平台[19]进行，所有数据可视化使用ggplot2包[27]进行。我们在dgof包中使用了Kolmogorov-Smirnov测试的实现，因为它能够正确地处理离散变量，例如作用域计数[2]。

1.2.3.5代码的可用性

这里提供的所有用于生成和分析数据的代码都是开源的、公开可用的。这段代码大量使用了Empirical库，它也是开源的，公开可用的。

1.3结果与讨论

使用MAP-ELITES的程序的范围计数和指令熵值分布与使用其他选择方案演进的程序有显著差异(Kolomogorov-Smirnov检验，p < 0.0001)。1.1和1.2,每个指标的范围与MAP-Elites程序进化比范围更广泛的项目发展下其他选择方法(除了在某些情况下的随机漂移,这实际上是不受程序的要求解决这个问题)。虽然这一结果并不令人惊讶，但它证实了使用地图精英作为探索GP表现的工具能够提供我们无法通过其他方式获得的信息。

1.4结论

我们已经演示了使用地图精英作为探索简单线性GP表示的工具。通过为与程序架构的各个方面相对应的地图精英选择表型轴，我们可以建立一种直觉，了解GP表示的不同特征与跨越各种问题的进化过程的相关性如何。随着GP社区的不断发展，这些分析类型非常重要。

在本研究中，我们将map -精英表型轴的选择限制在指令熵和模块使用;然而，有许多可能的信息轴。此外，地图精英并不局限于两个坐标轴。我们可以选择任意数量的特征来定义坐标轴，这样我们就可以探索在给定问题的背景下，GP表示有多少个不同的方面相互作用。在这种情况下，我们可以使用许多可能的度量标准。

在遗传程序设计中，评估基因组指令(或基于树或基于图的GP上下文中的操作)的组成是有意义的。我们可以评估基因组中所有指令的组成，或者只计算那些实际执行的指令。这样的分析可以使用以下指标来执行:

•程序中指令总数(长度)

•程序中惟一指令类型的总数

•指令熵(如本文所使用)

•指定指令类型被使用的次数

•指令类型使用次数的熵。

•指令[3]的平均有效依赖距离

我们可能还会关心程序架构的抽象属性。例如，有许多与模块化相关的量，度量这些量可能是有用的，特别是考虑到模块化被认为可以促进可演化性这一事实。其中最简单的方法是使用诸如[14]中描述的物理和功能模块度量来度量程序的模块性。在模块容易识别的情况下，我们可以通过以下指标进一步探测:

•给定的单个内存位置被引用/访问的次数

•所使用内存的模块化。

还有很多潜在的有用的度量标准将取决于所使用的遗传编程表示的具体情况。例如，树和基于图的程序(例如，笛卡尔遗传编程)可以用描述其拓扑结构的度量来评估。另一方面，具有线性基因组的表示法可以从生物学中借鉴各种有用的指标。

我们已经证明，使用基于程序架构的具有表型轴的地图精英可以阐明程序架构上的约束，这些约束是我们通过检查由传统选择操作符生成的程序而不知道的。理解这些限制可以帮助我们理解为什么某些遗传编程表示在某些情况下或多或少地成功，这是[17]领域长期发展的一个重要目标。因此，我们期望这里提出的方法将是我们用于研究遗传规划的工具包的一个有用的补充。

1. 进化自助算法

2.1引言

在遗传算法(GA)和人工智能(AI)领域中，各种各样的计算基板具有找到解决各种各样问题的能力。研究已经专门针对不同的计算基板，每个excel在不同的问题领域。例如,人工神经网络(ANN)[28]已被证明有效的分类、遗传程序(我们是指数学基于树的遗传编程和将缩写GP)[18]通常被用来找到适合数据复杂的方程式,神经增加拓扑的演化(整洁)[35][7]善于机器人控制问题,和马尔可夫大脑(MB)(8、12、21)是用来测试的假设关于进化行为[25](在许多其他的例子)。鉴于广泛的问题和大量的计算基板，遗传算法和人工智能的实践者面临的困难是，每个新问题都需要评估，以找到合适的计算基板和特定的参数调整，以实现最佳结果。

提出了将不同的计算基板结合起来的方法。AutoML[27,36]是一种为给定的问题选择最合适的计算基板，然后在该基板上生成解的方法。另一种复合方法是“专家混合”概念，允许从一组异构的子网络构建人工神经网络，最初由Jacobs等人首创，类似于最近的工作[31]。这些方法从现有的基质中选择或创建一个现有基质的网络。

在本文中，我们提出了一种复合方法，该方法从各种已知的计算基板中借用元素，并使用它们创建异构遗传算法，允许来自不同基板的组件之间直接进行低级集成。根据没有免费午餐定理[29,39,40,42]，我们称这种方法为自助餐方法，该定理大致“表明任何两种优化算法在所有可能问题上的性能平均时是等价的”[41]。在本文中，我们从四个计算基板中选择组件，并将这些组件与MB框架相结合，构造了一种可能的Buffet方法的实现。每一种底物之所以被选中，是因为它在某些领域被观察到是成功的。我们将显示一个MB包含广泛的异构计算元素从其他系统(GP,整洁、人工神经网络和MB)可以结合这些不同的系统的优势,导致自动发现高质量的能力,经常混合,遭受个人系统解决方案,同时避免陷阱。

2.2方法

我们将使用需要定义和/或说明的术语。当我们谈到“计算基质”时，我们指的是一种特定计算方法的定义。在这个意义上，ANN、MB等都定义了计算基板，但生物大脑也是如此(尽管我们不建议将生物神经元整合成一种buffet方法…)。然而)。同样重要的是，当我们谈论ANN、GP和NEAT时，我们谈论的是特定的实现(通常是初始的或规范的实现)。特别地，当我们使用GP时，我们指的是基于树的数学遗传规划，并绝不放弃研究其他形式遗传规划的大量工作。我们还想指出，我们在本文中以两种方式使用马尔可夫大脑。首先，MB被用作底层方法，允许来自不同计算基板的元素之间的互连。其次，我们将使用自第一次提出MB以来一直在使用的两种MB门。在讨论第一种意义上的马尔科夫大脑时，我们将使用MB，在第二种意义上我们将使用CMB(正则马尔科夫大脑)。

每一个计算基板都指定了唯一的子元件行为，子元件相互连接的方式，基板可用的动作，输入如何被接收，输出如何被传递，以及基板如何存储内部状态以允许记忆和重复。例如，GP由树状的节点构成，而ANN则有固定的分层拓扑结构。

CMB逻辑门在数字输入上工作，而不是在连续值上，如NEAT使用的那些。因此，不可能创建一个将所有系统中未修改的元素集成在一起的系统。相反，我们确定了每个系统的基本特性，并设计出一种方法，将这些特性整合到一个新系统中。

2.2.1马尔可夫模型（MB）

马尔可夫大脑描述了由三个主要元素组成的计算基板:节点、门和线路。节点只是值(输入、输出或隐藏/循环)。门是在节点之间执行计算的逻辑单元。导线将输入节点连接到门，门连接到输出节点。当一个MB被执行时，它的所有门都并行处理。

如果多个门输出连接到同一个输出节点，则对门输出值进行求和(其他方法，如覆盖和平均已经测试过，但它们的讨论超出了本文的范围)。在许多情况下，MB用于二进制输入并产生二进制输出，在这些情况下，输出值离散为1，如果值>为0，否则为0。

通常，MB用于寻找需要多次更新的问题的解决方案(如机器人的导航)。每次更新都设置输入(将传感器状态转换为输入节点)，执行MB，然后根据输出节点(即输出节点用于控制电机)的状态更新“世界”。更新之间的内存是通过隐藏节点实现的。这些额外的节点是通过在输入和输出缓冲区中保留额外的空间而添加的。每次执行MB后，写入输出隐藏节点的值都会被复制到输入隐藏节点。在一些MB的配置中，保留额外的输入节点，以便输出节点的值可以以与隐藏节点相同的方式复制，从而提供对最后输出的直接访问。

由于MB体系结构只有一层，所以输出值仅限于仅使用单门执行或此类执行的总和的操作。这个限制可以通过允许隐藏节点和使用一组输入多次执行MB来克服。更详细的节点和门的拓扑结构是可能的。例如，如果输出节点可以作为门的输入，那么一个门的输出可以在一个MB的执行中被另一个门访问。当然，在这种串行配置中，门不能并行运行，必须建立一个门的执行顺序(通常作为在建造门时从基因组中提取的附加信息)。

输入和输出节点的数量由任务决定，隐藏节点的数量由用户设置。门和线最初是随机产生的，但随后受到选择、复制和突变的影响(见下面的编码方法)。

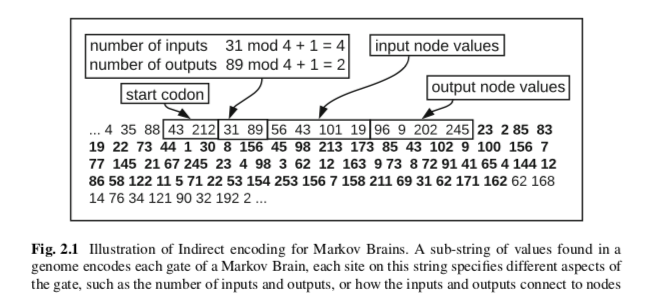
门可以有任意数量的输入和输出。基本门类型是一个确定性逻辑门，使用1到4个输入，将输入转换为位(使用上面提到的离散化函数)，然后从遗传决定的查找表提供1到4个输出。神经元门是一种更复杂的门类型，它接受一个或多个输入，将这些输入相加，如果输入总数超过可进化阈值，则提供可进化的输出值，如果不超过可进化阈值，则提供0。神经元门的累加值可以持续多次更新，为神经元门提供自己的本地记忆(除了隐藏的节点)。神经元门有额外的配置选项，例如，允许他们交互地改变他们的阈值。其他已经被探索的门类型包括计数器、计时器[13]、数学运算和反馈[32]。在这里，我们的目的不是描述所有可能的(甚至是存在的)门类型，而是要说明可能行为的范围不受限制，甚至可能包括嵌套的MB网络。添加新的门类型只需要门的内部过程(门更新)的实现以及构造和/或突变行为(取决于编码类型)。有关马尔可夫大脑的更详细描述，请参阅[12]。

事实上，包含在门中的逻辑可以定义任何计算操作。由于MB门的I/O标准化，无论内部的计算过程如何，任何门类型的集合都是兼容的。MB[12]的模块化互操作结构为创建采用多源元素的异构计算基板奠定了基础，这就是自助餐方法的由来。

有许多过程可以用来构建MB。虽然构造方法可能会影响演化性，但它可以从MB基底中单独考虑。为了产生更稳健的结果(即确保编码方法对结果不是关键的)，我们使用遗传编码方法和直接编码方法复制了所有的实验。

2.2.2基因编码方式

遗传编码使用基因组(一串数字)和翻译方法。预定义的起始密码子(数字的子串，例如‘43,212’)的出现识别了定义门的区域(基因)。每个门都与一个特定的起始密码子相关联。开始密码子之后的序列提供了定义该门的功能和它如何连接的所需信息。因此，基因组中每一个“43,212”的序列在线性读读过程中都会启动一个“门”(与“43,212”相关的类型)的后续翻译。注意，这允许重叠门基因。由于每个门类型需要不同的数据来构建它，因此必须从开始密码子之后的基因组中提取的信息将是不同的，并且必须由门类型定义。通过在翻译方法中添加或删除门类型的起始密码子，可以允许或不允许门类型。



例如，考虑图2.1中的基因组子串。如果这是一个基因组的一部分被翻译成一个MB，第一件事会发生的是' 43212 '子串将被定位。碰巧，' 43212 '是一个确定性门的起始密码子。接下来的两个值31和89将用于确定输入和输出的数量。因为确定性门有1到4个输入和1到4个输出，这些值都将用((value mod 4) + 1)处理;产生4个输入和2个输出。接下来的8个值决定了输入和输出地址。这个门将使用所有4个输入地址值，但只使用前2个输出地址值。由于突变可能改变输入或输出的数量，代表第3和第4输入的额外基因组位置是未读的，所以这样的突变不会导致帧移位。为了处理输入和输出地址值，输入基因组值被输入节点数(包括隐藏节点)调制，输出值被输出节点数(包括隐藏节点)调制。下面的64个值(图中的粗体文本)被2修改，为查找表生成64个二进制值。因为这个门只有4个输入和2个输出，所以这些查找表的值将不会被使用，但它们是未读的(像输入和输出地址值)，以避免在突变的情况下帧移位，这将改变输入或输出的数量。

复制和变异是简单的在使用这种形式的基因编码基因复制,应用随机突变,然后产生的基因组是翻译成一个新的MB。我们允许点突变(随机改变基因组的一个随机的网站),复制突变(部分基因组选择和复制到另一个位置)和删除突变(基因组的一部分被删除)。突变率是由用户确定的，并定义为每个位点的百分率。有性繁殖是通过亲本基因组之间的杂交实现的(尽管本文中的所有实验都是无性的)。

2.2.3直接编码

本文采用的直接编码方法是利用随机生成的基因组生成初始MB群体，并利用上述方法对基因组进行翻译，确保直接编码或间接编码的实验具有相同的起始条件。但此后，有机体通过复制MB并直接对MB的每个组成部分进行突变来进行繁殖和突变。这个方法增加了根据每个门类型的唯一结构为每个门类型指定突变操作符的要求。此外，必须明确地确定每个可能突变的突变率。本文的直接编码方法允许突变，可以添加一个新的随机生成门，复制一个现有的随机选择门，删除一个现有的随机选择门，改变门的输入或输出线，改变门的内部参数。

2.2.4多步骤的功能

MBs在单个更新中并行执行所有的门。如果一个计算顺序需要多个门的参与，那么必须有多个更新。如果一个任务需要多次更新，并且MB执行多次更新(例如一个重复输入/更新/输出循环的导航任务)，那么随着时间的推移，就会发生多步骤的过程。另一方面，有些任务会提出问题，并期待得到答案，或者时间紧迫。在这些情况下，我们允许MB在设置输入和读取输出之间多次更新。这在概念上类似于允许evolution使用Jordan循环体系结构、Elman循环体系结构和可演化连接的部分，如NEAT[9,16,35]。

2.2.5门类型

在本文中，我们使用了四种架构:规范马尔可夫脑(CMB)、GP、ANN和NEAT。我们之所以选择这些方法，是因为它们在不同的问题领域都表现出色。其他系统的元素，包括神经图灵机[22]，超整洁的[34]，和POMDP[17]也被考虑，可以在以后加入。

我们的目的不是比较这些不同的计算基板，因为我们在MB框架中重新实现它们，这是不可能的，也没有意义的，而是使用这些架构进行研究。

为了表示MB，我们选择了两种门类型:确定性门和概率门，它们是最早使用的也是最常用的门。当我们提到这些门类型的使用时，我们将使用缩写CMB(即Canonical Markov Brain)而不是MB，以避免混淆。这些门有1-4个输入和1-4个输出。update函数是一个查找表，它将输入映射到输出。在概率门中，每个输入模式都可以产生任何输出模式。对于每个可能的输入模式，每个可能的输出模式都被分配一个概率。要确定给定输入产生的输出，需要在门更新期间生成一个随机数。

从GP那里，我们借鉴了一元和二元数学运算的思想。GP门接受1或2个输入，然后执行生成单个输出的操作(见表2.1)。

在人工神经网络中，我们采用传递函数(加权输入之和)和双曲正切作为阈值函数。这些ANN门有1-4个输入作为输入层，1-4个输出作为输出层，相当于一个没有隐含层的小ANN。这种神经网络门的具体函数由一个权矩阵控制。

从NEAT我们借用了更复杂的加权和积分方法[33]。这些门是ANN和GP的混合体，它们接受多个输入，施加权重，将它们聚合(乘积、和、最大值、最小值或平均值)，然后通过数学运算传递它们。这些门有1-2个输入，一个输出(具体使用的操作符见表2.2)。

2.2.6任务

我们选择了一系列任务，其中包括上面描述的每种gate类型都已证明其能力的示例。这里的目的不是将这些门类型或Buffet方法作为一个整体进行基准测试，而是演示Buffet方法允许进化来利用不同门类型的计算能力，平均而言，这比使用单一门类型产生更好的结果。此外，我们的结果表明，进化常常找到由门类型的组合组成的解决方案。

表格

描述已自动生成

2.2.6.1 Xor

对于这个任务，给出两个二进制输入，如果逻辑异或操作被正确地计算为[15]，则适合度被授予。对于一个MB来说，这是一个非常简单的任务，因为初始人口是由随机生成的逻辑门组成的，而逻辑门很可能实现这个功能。然而，通过进化神经网络来解决该任务并非易事，该任务以前曾被用作NEAT的基准示例[15,37]。适应度(性能)的评估是通过一次提供100对二进制输入，并将代理输出与预期输出进行比较。正确的答案被统计，结果是0到100之间的适应度。每个代理有10个更新，允许在评估输出之前执行多步计算。

2.2.6.2回归

我们为符号回归组织了一小组函数，对于遗传算法来说，不管用什么方法(数据未显示)，这些函数似乎都同样复杂。这里，我们展示了一个具有两个输入x1和x2的函数的结果:f (x1, x2) = (x1∗x2)∗(x2−x1)。代理的适应度是由这个函数和代理的响应(在−2.0和2.0之间随机输入值采样100次)之间的差决定的，这是求和和平方。每个代理有10个更新，允许多步计算。

图示

描述已自动生成

2.2.6.3倒立摆

这个任务[4]涉及平衡一辆小车上的垂直横梁，小车只能像在铁轨上一样向左或向右移动。每个药剂评估10次。横梁(摆)安装在小车的顶部，使其可以绕其安装点在1个垂直轴上自由旋转(概览见图2.2)。代理可以使小车向左或向右加速，并且在100次模拟更新期间记录钟摆在小车上方的时间。输入当前的钟摆Θ角,它的一阶导数Θ̇,车的位置x, ẋ当前加速度的车。对于每次模拟更新，agent经历8次多步计算，输出是cart加速−→F(限制在−1.0和1.0之间)。这个任务的输入和输出都是连续(浮点)值。代码是从openAI[26]移植过来的。

2.2.6.4价值判断

这项任务起源于心理学中的决策。agent面对两个有噪声信号时，必须识别较强的信号[19]。想象一下，有两盏闪烁的灯，你必须辨认出哪盏灯更亮。在这个任务中，每个代理有两个二进制输入和两个二进制输出。每次评估包含100个更新。在每次评估开始时，生成一个随机值0或1，以确定第一个或第二个输入更有可能。在每次更新中，更有可能的输入是1而其他输入是0的概率为55%，相反的概率为45%。对于前80个更新，将忽略输出。然后代理有20个更新来提供答案。

输出0、0或1,1将被忽略，但0、1和1,0将触发计算的结束。如果提供的输出是0,1，并且第二个输入更可能是1,0，或者第一个输入更可能是1,0，那么代理收到一个点。此测试重复100次，并对结果进行求和，结果的适应度值在0到100之间。注意:a)猜测会得到平均50分，b)完美的分数是不太可能的，因为眨眼概率0.55和0.45之间有微小的变化。

2.2.6.5 Block-Catching任务

这个任务涉及到一个嵌入的代理，它必须捕获小块或避免斜向代理[5]移动的大块。这是我们之前彻底研究过的一个任务[3,21,30]，它可以通过MBs和ann来解决(该任务的说明见图2.3)。这项任务并不简单，因为信息必须随着时间的推移进行整合。在这个任务中，个体在每个世界的更新中只能进行一次大脑更新，所以要想成功，他们就必须在出现新信息时整合和记忆信息。

2.2.6.6联想记忆

在联想学习任务中，智能体沿着预先定义的路径访问每一个独特的位置都会得到奖励。给代理一个输入，表示代理是否在路径的直线上。还提供了另外两个输入，一个是路径走向左边，另一个是路径走向右边。在agent访问路径上的每一个独特位置时，适应度都会增加，而在agent不在路径上时，适应度则会减少。在每个适应度评估之间，“左转”和“右转”的输入集合是随机的。没有提供指示路径结束的信息。在适应度评估的开始，agent首先应该发现标志和转向方向之间的映射，如果执行了这种行为，就会看起来像探索。然后代理可以利用该信息来正确地遵循路径，这看起来像一个开发阶段[11](图2.4)。最初的工作是在AVIDA[1]中进行的，这种特殊的扩展任务到联想学习是由Anselmo Pontes在尚未发表的论文工作中提出的。从先前的实验(数据未显示)中，我们知道MBs和ann都很适合解决这个问题。

2.2.6.7嘈杂的觅食

这个任务使用一个具身代理，它必须在一个离散的网格上搜索随机放置的食物;一旦食物被收集，代理人必须回到先前定义的家的位置来提高适应性。这种觅食之旅可以在代理人的一生中重复多次，每一次重复随机放置的食物被移到离家更远的地方。家庭和食物的位置是由一个灯塔标记的，可以从很远的地方被代理人看到。特工有八个探测器每个覆盖45◦弧，可以检测食物和家的信标。代理可以45◦左、右转弯或向前移动。额外的传感器会通知探员已经到达了家中的位置或可以找到食物的位置。每个特工有9900到10155个时间步骤来收集食物。生命周期的变化阻止代理部署重复的搜索模式(数据未显示)。适应度根据以下函数定义:

图示, 示意图

中度可信度描述已自动生成

2.2.6.8迷宫解决任务

在这个导航任务[8]中，agent会因为在由一系列平行的长墙组成的二元迷宫中导航而获得奖励，每个长墙都包含沿着墙壁的某个地方的一扇门。当agent通过一扇门时，如果下一扇门在当前门的右侧，则接收输入(1);如果下一扇门不在当前门的右侧，则接收输入(0)。agent只能执行“向前一步”、“向左一步”和“向右一步”三种操作。这个任务要求代理识别零星的信号并记住它们(至少直到隔壁)，以便有效地导航。

2.2.7实验参数

对于每种可能的大脑组合(CMB、GP、NEAT、ANN和buffet)、环境(XOR、符号回归、倒立摆、价值判断、浆果世界、时空整合、迷宫、联想学习和噪声觅食)和编码(直接和间接)，我们调查了每种条件的100个重复。每个条件都是在第一代中随机生成10000种药剂(以增加初始可行药剂的可能性)。此后，种群规模减少到100个个体，进化持续了5000代。在每一代中，轮盘赌轮选择被用来选择父母，通过无性繁殖，变异的后代被产生。我们期望，我们的结果也应该适用于其他搜索方法，如地图精英[24]，新奇搜索[20]，复杂登山者[10]。

所有实验都是使用MABE框架[6]实现的。MABE是一个通用的数字进化框架，旨在允许模块的任意组合，以构建基于agent的进化实验。在这里，我们利用了MABE在实验之间“交换”世界模块(适应度函数)的能力，同时保留计算基质、基因组、突变操作和选择方法不变。

2.3结果

虽然大多数大脑和环境的组合都表现得很好，但也有一些组合没有找到解决方案或表现得不太理想(见图2.6)。例如，NEAT, GP和ann在浆果世界和觅食环境中挣扎，CMB无法在分配的时间内解决钟摆任务。Buffet方法能够很好地解决所有的任务。

对比CMB、GP、ANN、NEAT和Buffet方法，我们发现Buffet方法无论直接编码还是间接编码，总体上都是将种群进化到更高的适应度(见图2.7)，但在某些情况下并不是最好的。特别是，ann在倒立摆任务中表现得更好，CMBs在联想记忆任务中表现得更好。

为了研究进化选择哪些组件，我们重构了CMB门、NEAT门、GP门和ANN门在进化时间内的门使用情况。看到Fig.2.8。在一些环境中(符号回归和倒立摆)，ANN门是主要使用，而在XOR和Berry世界，CMB门占主导地位。在所有其他环境中，我们发现不止一种门类型会轻微偏向于一种类型或另一种类型(取决于任务)。检查个体的大脑(数据未显示)证实进化的大脑是由不同的门类型组成的。这表明，Buffet方法不仅允许进化选择单一的最优门类型，而且还产生由不同门类型组成的异构大脑。

2.4讨论与结论

我们发现，Buffet方法在所有任务中都表现良好，而每一个子集(CMB, NEAT, GP, ANN)至少有一个任务失败，在其他任务上表现不佳。

虽然我们不能确定为什么不同的计算基板在某些任务中会遇到困难，但值得注意的是，对于NEAT、GP和ANN来说，有问题的两个任务涉及定向导航。我们知道CMBs的困难摆任务的结果是,MB盖茨不能产生负的输出(因为我们选择代表招商银行与确定性和概率盖茨二进制和非负),因此只能加速马车在一个方向上。我们本可以通过改变倒立摆任务的输出含义来解决该任务中CMBs的负数问题。我们可以将输入离散化为一串位元而不是力，我们可以使用两个二进制输出，其中00和11都没有意义，01或10表示向左或向右的加速度。我们之所以没有重新实现倒立摆任务的输入和输出，使非连续值基板能够解决它，是为了强调并不是所有计算基板都能解决所有问题。我们的方法允许我们测试Buffet方法是否能够使用所提供的元素来发现问题的解决方案，前提是所提供的元素中至少有一部分可以解决问题。buffet方法的优点之一是它允许将显著不同的计算基板组合在一起。在这种情况下，衬底在输入和输出细节上有不同的限制。之前的工作已经表明，任务的表现方式可以影响进化的结果[2]。使用Buffet方法，我们可以包括任务，而不需要考虑任务如何，甚至是否会与基质接触。

考虑到Buffet方法可以访问每个计算基板的所有元素，值得注意的是它并不总是能找到最优解。例如，在钟摆任务上，只使用ANN门的MB优于Buffet方法。为什么Buffet方法不简单地抛弃所有其他门而保留安门?同样的逻辑适用于关联任务，单独使用MB门产生比Buffet方法更好的结果。我们认为这种影响可能与历史偶然性有关。也就是说，如果发现了一个使用次最优门类型提供一些适应性的策略，那么这个门和策略可能已经被“锁定”，禁止发现更优的解决方案。但这只是猜测，需要更多的调查。

2.5未来的工作

在这里，我们将ANN、NEAT和GP门合并到MB基板中，但是将概率和确定性MB逻辑门和ANN加权阈值元素整合到GP或NEAT中还有待研究。这个想法并不完全是新奇的。例如，笛卡尔GP的一些实现包括二进制逻辑元素[23]。未来对集成计算基板的探索，允许任意集成的方法，以及测试这些新兴系统的方法，为研究和开发提供了充足的机会。

这里介绍的工作表明,找到正确的计算衬底的典型方法对于一个给定的问题是次优的进化的自助方法不仅可用于发现计算衬底对给定的问题是最优的,但是可以生成新的混合动力系统在自动方式。

我们绝不是坚持每个人都应该放弃他们正在做的事情来研究Buffet方法，我们希望工作继续探索不同的领域，这样每当一个新想法表现得很好(即使领域很窄)，这个想法就可以被整合到Buffet中。

我们没有探讨的一个领域是基于gate使用情况的任务分类。该方法可以应用于更大数量的任务，并可以比较访问某个门子集时获得最优解的时间。或者，可以比较所有门都提供时用来组成解的门的分布。这样的概况可以提供一种客观的任务分类方法。

最后，我们发现Buffet方法产生了由不同组分组成的新的异质解。人们不禁要问，为什么从来没有打算一起工作的组件，会突然形成有功能的计算机器。我们将在未来探索这些混合基质。

1. 通过复杂的程序图进行视觉强化学习

3.1引言

视觉强化学习是将强化学习算法直接应用于来自摄像机或视频源的帧(像素)数据。因此，学习主体能够比以前更直接地与环境交互，也就是说，不存在关于哪些特征是有用/重要的先验决策，这可能会减少偏见的来源。到目前为止，这些方法已经被深度学习的结果所主导，这些结果成功地减少了源图像所需的预处理量(通常是通过连续图像平均来降低采样)，同时证明了提高人类[28]性能的能力。

深度学习之所以能够在视觉强化学习任务中提供最先进的性能，部分原因是由于最初的高维感官输入(像素)被编码为有效的低维(er)表示的明确方式。通过高度模块化的层次结构(深度学习)找到合适的编码后，一个决策制定组件被同时训练来提供代理的策略(通常是一个多层感知器)。同样重要的是，使用共同的拓扑结构和超参数(游戏邦注:如49款雅达利游戏)来实现这些结果。自从Mnih等人的开创性报告以来，大多数研究都集中在改进深度学习中使用的强化学习的初始形式(这种形式有很多，参见[24])，或者提出进化深度学习体系结构不同部分的方法。后者意味着深度学习体系结构的复杂性将更紧密地反映任务的潜在复杂性，例如[29]。

在这项工作中，我们回顾了最近的一项结果，该结果采用了一种完全不同的方法，其中程序模块化扮演了核心角色;以下是复杂程序图(TPG)[12, 13, 15, 31]。因此，我们不是学习编码一个较低维度的表示，而是学习如何连接多个团队的程序(模块)。每个团队单独尝试基于低维策略做出决策，因此团队根据非常小的像素信息子集来分解任务。通过这种方式，团队的组成和团队之间的连接都是紧急的。最后，每个决策实际上只使用了代理团队的一小部分。因此，当一个代理对不同的状态做出反应时，团队的不同子集也会做出反应。所有这些结果在视觉强化学习任务下形成了一个非常有效的决策制定框架。

在下面，我们回顾了相关的工作(第3.2节)并描述了视觉强化学习(第3.3节)。第3.4节介绍了TPG框架的教程概述，然后介绍了在Atari电子游戏和VizDoom环境中使用TPG进行决策的案例研究(分别是第3.5节和第3.6节)。3.7节作了结论性讨论。

3.2相关工作

TPG强调程序之间相互关系的发展，最终导致(纠缠)图的出现。然而，这是对早期基因编程中团队隐喻研究的概括。考虑到这一点，我们将从这两个角度简要介绍相关工作。当然，还有许多其他有趣的发展涉及到图形和程序，例如笛卡尔GP[27]，基因调控网络[2]，和图形编程[1]。然而，这些发展倾向于使用图表来表达单个指令之间的相互关系，而TPG的贡献在于提供程序之间相互关系的紧急组织。

3.2.1进化图

Fogal是序列学习任务[8]有限状态机进化的先驱。然而，这些模型需要针对所有状态和输入的组合的转换规则，这限制了它们的可伸缩性。PADO出现于1996年，用[35]程序图来表示解决方案。该图有一个开始节点和结束节点。每个节点都有一个程序来操作自己的堆栈，并支持全局内存的索引。通过图的路径由与每个节点相关联的单个条件指令决定。从开始节点上的程序开始执行，直到超时或遇到完成节点。如果输入保持不变，那么似乎每个节点本地程序的状态都保留了。我们不清楚图中有多少是作为突发性属性开发的(例如，如果节点的数量是预先定义的/在整个进化过程中保持不变)，但在每次执行过程中，图中的大多数(如果不是全部的话)节点似乎都被访问了。

最近有人提出了遗传网络规划(GNP)，即预先声明固定数量的图节点，每个节点必须是两种“类型”中的一种:条件型或动作型[23]。节点的数量是固定的(在进化过程中是常量，对所有个体都是通用的)，以及有限的(特定于应用程序的)条件操作符和操作集。GNP在一个特定的开始节点初始化，然后允许执行最多不超过5 ' ticks '。每个节点类型都有一个特定的标记成本(条件1，操作5)，这限制了环境中每个状态访问图的数量。实际上，GA用于定义节点之间的连接性和相对于预定义的条件或操作库的节点类型。

增强拓扑的神经进化(NEAT)代表了一个开发任意拓扑的神经网络的框架，从感知器的总体开始(每个感知器最初完全连接到输入)。因此，基因型表达了不同类型节点(输入、输出和神经元)之间的连接图。此外，为了建立跨界[33]的上下文，NEAT还引入了一个连接的基因型标记方案。同样的标记方案构成了基因型多样性测量的基础，该测量在进化过程中保持种群中固定数量的生态位。该框架能够描述反复出现的连接，以及定义权重值。这个整洁的框架已经被广泛采用，甚至在雅达利视觉强化学习领域[9]下进行了基准测试。后来的发展，如HyperNEAT，集中于以一种特别紧凑的基因型表达非常大的神经元阵列，但结果是种群中所有成员都保留了固定数量的神经元[9]。

其他一些进化神经网络(因此也包括图)的框架已经被提出，包括通过笛卡尔GP[37]的神经进化和使用线性遗传编码来表达图[25]。迄今为止，这种方法还没有应用到视觉强化学习任务中。最近，DeepNEAT框架被提出为[26]，其中每个基因组节点代表一层，而不是单个神经元(就像NEAT那样)。具体来说，每个基因组是一个参数表，用于表征深度学习体系结构的各层，基因组的边缘表示各层之间的连接。通过图像分类和语言建模任务进行评估。

3.2.2多程序无图演进

遗传规划中的团队隐喻已经被提出，并根据所有同时操作的程序的集合来定义合作。早期的研究假设一个固定长度的基因组，因此程序的数量总是预先声明的，并且不会因进化过程[4]而改变。在其他情况下，假设是可变长度的基因组，但适应度必须在程序和团队的“级别”上指定，因此将应用基础限制在分类任务[36,40]。

为了避免这些问题，之前假设了一个共生框架，其中一个种群寻找有用的团队成员，另一个种群提供[20]程序的种群。此外，我们还采用了学习分类器系统的方法，即明确区分何时采取行动和采取什么行动的决定，或者基于出价的GP(图3.1)[19]。因此，给定一个基于竞价的GP个体团队，每个人的程序在给定任务当前状态s(t)的情况下执行。团队中产出最多的个人被认为“赢得”了建议其行动的权利。操作只是一个标量a，它是从特定于标量任务(原子)的操作集合a中取来的。因此，对于一个三类分类问题，a ={0,1,2}。每个基于出价的GP个体只能有一个单独的动作，这意味着程序上下文会根据一个

图示

描述已自动生成

3.3视觉强化学习

在下面，我们将假定视觉强化学习任务与离散的行动。例如，Atari (arcade)学习环境(ALE)[3]根据时刻t的帧缓冲区定义状态，t ={0,1,2，…，n}表示n + 1个连续帧的序列。2决策代理人的目标是为每一帧显示一个动作游戏分数最大化,即推迟到一些奖励标准是遇到结束,如代理游戏交互的最大数量(t =最高温度)或游戏状态的结束。在ALE下，可用的(原子)动作集合A，根据多达18个原子动作3定义，对应于操纵杆的8个离散方向的枚举

有或没有按钮“press”，4加上一个按钮单独按和“无动作”(NA)。因此，我们的兴趣在于找到一个能够直接根据视觉状态信息玩多款游戏的单一机器学习框架。没有尝试先验地识别适当的输入特征或将任务分解为一系列递增的更困难的训练场景。因此，学习代理必须根据人类玩家所能感知到的信息(尽管没有任何声音信息)来确定玩游戏的合适策略。

到目前为止，这种视觉强化学习任务已经被深度学习的发展所主导(见[24]的评论)。进化的方法已经被提出，需要先验信息来预处理原始帧信息到单独的“通道”[9]，或限制优化先验深度学习体系结构的参数[29,34]。我们还提出了一个GP公式，其中GP定义了一系列图像处理算子[10]。然后，GP处理的结果图像需要通过一组启发式解释，以确定行动。因此，这种方法仅限于可以设计适当启发式的游戏子集。最近，笛卡尔GP在ALE上得到了演示，尽管是在假设一个包含显式操作向量(即图像)数据[39]的操作符的大型指令集的情况下。

此前，TPG被证明在深度学习在玩[12]时表现不佳的20款ALE游戏中特别有效，然后在所有49款ALE游戏[15]中进行了基准测试。TPG也被证明有能力制定出一项政策来应对多个游戏项目[13,15]。最近，TPG在VizDoom第一人称射击游戏环境[31]下进行了演示，这表明TPG也可能比ALE下扩展到更大的状态空间，并在比通常在ALE中呈现的更高水平的部分可观测性下运行。

3.4 TPG图

TPG代表了一个将多个程序组织成结构的框架，这样它们就可以以高度模块化的方式解决一些更大的任务。我们的出发点是一个共生进化框架(图3.1a)，包括:(1)一个定义TPG图中的节点的节点种群和(2)一个程序种群。节点定义哪些程序将协作以在该特定节点上做出决策。程序总体根据程序、p和原子作用a∈a来定义每个个体(图3.1b)。两个种群在一种共生关系下共同进化，节点种群搜索“好的”模块(程序团队)，而程序种群专注于寻找“有用的”程序。

在初始人口中，所有节点都由2和ω(基于bid的)程序组成，用动作a随机初始化，从原子动作a集合中以均匀概率分配，在以下约束条件下:

•与同一节点关联的程序集合中必须至少存在两个不同的操作。

•每个节点必须有一个唯一的程序补码。

许多的地图

描述已自动生成

这样的出发点意味着:(1)所有图最初只有一个节点(图3.2a)，(2)同一个程序可以出现在多个节点(图3.1a)。

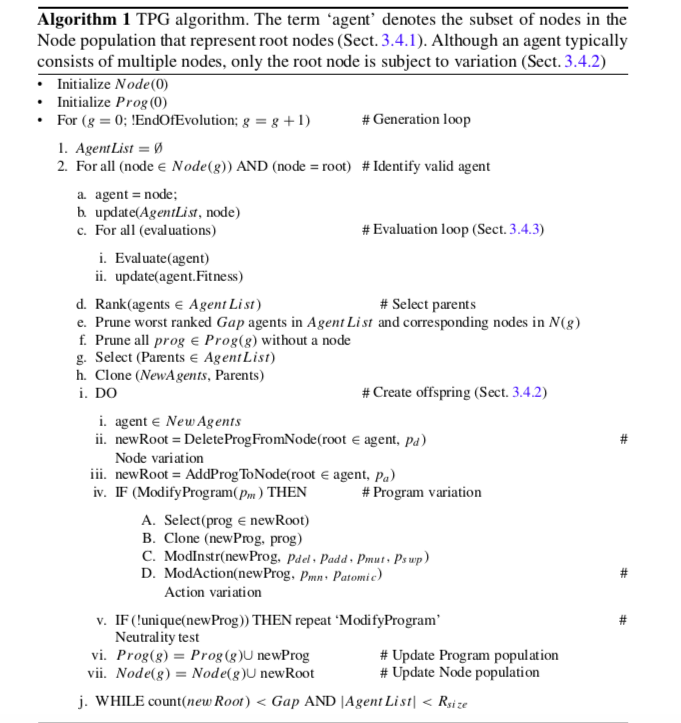
适应度评估后，对agent进行排序(步骤2d，算法1)，并从节点种群中删除性能最差的Gap(步骤2e)。变异运算符将抽样直到新因子引入到节点种群中为止(第3.4.2节)。除了精英主义，采用繁育模式并没有什么特别的意义。

在这里值得注意的是:

•虽然假设有两个人口框架，但适用性只是明确的

•在删除表现最差的代理之后，对项目成员进行测试。如果任何程序与节点填充中幸存的个体没有关联，则删除该程序(步骤2f)。这意味着程序总体的大小实际上是作为选择和变异操作符的函数波动的。

•节点人口中的节点可能是根节点或者是一个图的内部节点。很明显，代理的数量等于根节点的数量。因此，通常情况下，代理的数量小于节点的大小，精确的组成也是一个突发性的特性。考虑到这一点，引入一个测试来确保在每一代(步骤2j)中维护最小数量的代理(Rsize)。



在介绍TPG算法第3.4.1节的进一步细节之前，我们对共生表示如何从程序团队(每个节点都是一个独特的团队)将解决方案发展成一个复杂的图提供了一些直观的认识。第3.4.2节将定义变异运算符，并将其函数与整个TPG算法(算法1)联系起来。最后，第3.4.3节介绍了在给定帧缓冲区输入的情况下如何评估TPG代理。

3.4.1发展周期

图3.1在初始化时，每个节点(节点总体)从程序总体中识别出一个唯一的程序子集。我们可以通过忽略与两个群体的关系(基因型属性)，而仅仅专注于代理的表示(最初每个节点都是代理)，即表型属性，使图上的连接更加明显。

图3.2a说明了初始人口由三个代理a、b、c组成的情况，每个代理仅由单个节点组成。agent b和agent c由三个程序组成，agent a只有2个。以代理a为例，由两个程序1和3组成。弧线方向总是从程序有团队成员的节点开始，并在相应的动作结束。考虑到这是初始的总体，操作只能是原子操作，图3.2a，因此节点总体的所有成员都是根节点。每个根节点代表计算/执行开始的节点(章节3.4.3)，因此每个代理可能只有一个根节点。

图3.2b说明了修改程序“5”的动作所产生的效果。动作修改可以导致选择不同的原子动作(a∈a)，也可以导致程序指向另一个节点(a∈N(g))。当程序“5”被修改时，它的标识符改变(变为“9”)，以反映原来的程序“5”可能仍然存在于其他地方，即被一个不同的节点所使用(因此在应用变异操作符之前使用克隆，第3.4.2节)。节点' c '现在不再代表根节点，因为它是节点' b '的子节点。因此，代理的数量实际上减少了。

在图3.2c中，图3.2b中的节点“a”作为父节点进行了两次采样，并在两种情况下克隆了节点“d”和“e”。节点“e”继承了节点“a”的程序，但其原子操作发生了变化。相反，节点“d”保留了一个程序(“3”)不变，但也获得了程序“6”和一个新的程序“10”。由于程序“10”的动作是指向节点“a”的指针，节点“a”不再是根节点。至此，我们有三种代理:⟨b: c⟩，⟨d: a⟩，⟨e:∅⟩，其中，⟨x: y, z⟩表示具有根节点“x”和非根节点“y”和“z”的代理“x”。

图3.2d说明了一个具有三个节点的代理出现的点。在本例中，节点“b”获得了第4个程序(“14”)，其动作标识节点“a”。此外，节点“e”也添加了一个程序(“13”)，在这种情况下，该程序也碰巧使用了节点“a”作为其动作。我们现在有一个3个节点代理，⟨b: a⟩，c⟩，以及两个具有2个节点的代理，⟨d: a⟩，⟨e: a⟩。请注意，为了清楚起见，我们把上面的评论集中在作为选择和变异函数引入的后代代理上。父母将被保留下来，并与后代竞争生存的权利。

3.4.2变异

在确定父代(第2g步，算法1)之后，每个父代都被克隆。更具体地说，只有每个父节点的根节点会被克隆和变异(在初始化时，所有节点都是根节点)。这意味着更简单的父代和更复杂的后代之间的竞争是强制性的。因此，为了生存，更复杂的后代必须比(更简单的)父母表现得更好。

第一组变异操作符在克隆的(根)节点上操作，其形式如下:

删除(Step 2(i)ii)或增加(Step 2(i)iii)对计划(pa, pd)的引用，

•修改节点(pm)中当前的程序

第二组变异算子作用于与根节点子代相关的程序(仅在pm测试为真时调用，步骤2(i)iv)，在这种情况下，受影响的程序首先被克隆，然后再对克隆的程序应用变异算子:

删除或添加一个指令(pdel,padd)，

•改变指令(pmut)，

•在同一个程序中交换两条指令(pswp)，或者

•在团队(pmn)内修改程序的操作。

如果pmn测试为真，则应用一个附加的测试，patomic，它建立了动作改变的类型(步骤2(i)ivD)。因此，对于克隆程序的动作patomic true, ai是从原子动作集合ai∈A中选择的，而对于patomic false，新的动作是指向第g代节点填充中的任何节点的指针，orai∈N(g)。

在动作变化过程中强制执行一个约束。当程序插入到相应的节点中时，必须至少有一个原子操作出现在该节点本地的程序子集中。这个属性将在以后被用来保证在计算过程中不会产生无限循环，也就是说，所有的状态都将产生一个原子动作(章节3.4.3)。

最后，考虑到适应度评估的高成本，我们评估方案的唯一性(步骤2(i)v)。为此，将保留在适应度评估下执行的50个最后状态观察的集合。然后测试每个新程序，以确定新程序的出价值是否至少与Prog(g)中的任何其他程序的τ值不同。如果没有，则重新应用程序修改的变异操作符。实际上，这代表了利用竞价行为的最小阈值来检验变异运营商的中立性。

3.4.3评估

下面我们假设状态信息s(t)对应帧缓冲区的当前内容。代理的评估总是从根节点开始。此后，通过代理图的路径是动态的，通常导致在确定一个动作之前，只评估代理程序图的一小部分。与当前的深度学习或神经进化方法相比，这使得TPG在评估方面异常高效(这两种方法都会评估所有拓扑对每个决策的贡献)。

一个节点的计算是一个两步过程，在这个过程中，只执行与该节点相关的程序的子集。在这个子集中，只有出价最大的程序“赢得”建议其相应行动的权利(前提是它没有“标记”，见下文)。因此，计算一个节点可以确定一个特定的弧。然后有两种情况，要么arc引用一个原子动作，要么它引用图中的另一个节点。我们处理这些案件的方式如下:

•动作是原子的:这表示代理的决策。然后，世界的状态将得到更新，评价的过程将重复。

•动作是非原子的:弧被标记，节点被指向，如上所述。环境状态s(t)不变。

在标记的弧“赢得”节点计算的特殊情况下，这意味着有问题的节点之前已经被访问过，并且检测到一个循环。通过在节点评估时从候选程序集中删除这条弧线，然后返回中标弧线，从而打破循环。如果这也被标记，删除标记的弧线并在这个节点上选择下一个可用的中标的过程将重复。因为每个节点必须至少有一个原子动作，所以总有办法跳出循环，例如每次访问同一个节点时，它必须使用不同的弧线“退出”。

图3.3说明了假设的TPG代理和状态的评估过程。Subplot (a)表示与代理的根节点(程序0、2、5、9)相关联的程序的执行。注意，每个程序都可以自由地对状态空间(帧)的任何部分建立索引，增加了分解任务的能力。程序9中标了，并确定了下一个进行评估的节点(这条弧线也被标记了)，子图(b)。这个节点只有两条(出去的)弧线，分别对应程序3和程序7。执行这两个项目可能意味着完全不同的状态信息利用,次要情节(c),中标项目被确定为项目7原子动作,所以评估完成,次要情节(d)。这一行动会导致游戏状态改变对《阿凡达》的假设这个动作,因此游戏状态是先进一步,s (t)→s (t + 1)。当然,如果s(t + 1)对应博弈结束条件，则完成agent的适应度评估。否则，所有标记的弧将被重置，并从根节点重新开始代理评估。

图示

描述已自动生成

3.5案例研究:Arcade环境

第3.3节提供了街机学习环境(ALE)的背景[3,24]，它代表了视觉强化学习最广泛使用的基准之一。基本目标是在一个共同的参数化下从ALE库返回一个能够玩不同游戏的代理。提供给代理的唯一信息是来自帧缓冲区的视觉信息，以及特定于游戏标题的操纵杆动作子集。我们还注意到啤酒提供了各种来源的不确定性包括:随机抽样框架提供的代理,粘性的行动,国米帧精灵所描述的变化,并根据游戏标题,部分可观测性,也就是第一人称视角(见[24]这些属性的讨论)。

深度学习采用的方法假设:(1)“帧堆叠”，一种对物体运动进行编码的机制，(2)屏幕向下采样(84×84像素)和灰度像素，(3)关闭“粘性动作”[24,28]。作为这一比较的基础，我们将假设13个标题，这些标题是最近在ALE下对视觉强化学习的进化计算方法的评估的共同内容。具体来说，Salimans等人描述了一种基于进化策略的方法，用于识别先验指定的深度学习体系结构[29]的权值。诸如此类，使用遗传算法提供了一个非常紧凑的深度学习体系结构描述，该体系结构由400万个权重[34]组成。在这两种情况下，作者强调进化计算方法比原始的深度学习体系结构DQN[28]训练更快(在广泛的GPU支持下)，同时提供竞争性的代理策略。

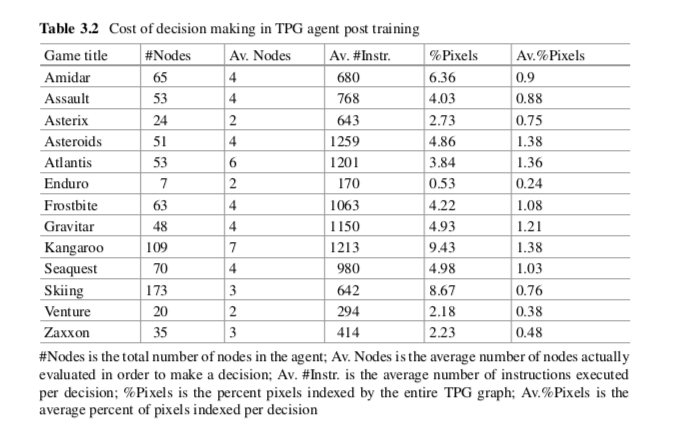
TPG的初步结果啤酒采用帧的预处理程序仅限于8潜在的每个像素颜色值(即SECAM制式的颜色编码,[3]),每一帧是量子化的5倍,导致1344十进制状态变量的输入空间范围在0 - 255(12、13)。在这里报告的结果中，没有尝试降低初始状态信息的维数。这意味着TPG代理所认为的屏幕包括所有210×160 = 33600像素，每个像素有128个可能的颜色值(即NTSC颜色编码，[3])。该指令集与之前的研究将GP团队应用于[21]分类任务时采用的8条指令相同，即没有使用应用/图像特定的操作符。图3.1b是使用该指令集的典型程序。根据之前的TPG结果，也存在“粘性行为”，这意味着TPG代理所经历的ALE环境是随机[24]。

表3.1总结了13个ALE游戏中每个代理的平均得分(在无操作游戏初始化[28]中平均超过200次评估)。显然，特定算法在不同的游戏子集上表现得尤为出色。同样明显的是，DQN同时拥有最多数量的排名1的游戏(最好的)和最多数量的排名4的游戏(最差的)。相反，TPG似乎在这组游戏中表现得最为一致，只有一个排名最差的游戏(亚特兰蒂斯)和所有游戏中最好的平均排名。简而言之，TPG在代理政策的质量方面具有竞争力。此外，TPG提供这些结果时，是在原始ALE分辨率下运行，没有框架下注。

表格

描述已自动生成

表3.2总结了TPG解决方案的复杂性，为了比较，GA个体描述了一个具有400万个权重[34]的深度学习体系结构。此外，由于深度学习体系结构中存在卷积运算，每个决策实际执行的计算数量要比这多得多。很明显，TPG中最坏情况下的决策成本(≈1200条指令)至少比深度学习中经历的决策成本少3个数量级(最多可少5个数量级)。简而言之，TPG可以在不需要gpu等专门硬件支持的情况下发展视觉强化学习任务的解决方案，因为它通过分而治之的共同进化过程明确地分解了任务。



一般来说，TPG策略对可用帧像素的覆盖非常稀疏，通常少于5%，而每个决策需要的帧像素不到2%，见表3.2。这反映了一个事实，雅达利电子游戏(以及一般的视觉信息)有许多冗余信息。特别是，屏幕上的大部分内容是为娱乐价值而设计的，而不是对决策很重要。此外，虽然重要的游戏实体通常大于单个像素，但代理可能只需要感知非常小的像素，以便检测和响应这些实体。自动扩展这些环境属性的能力有助于提高生成策略的整体效率。

3.6讨论

很少有机器学习范式能够在多个抽象层次上自动构建模块化拓扑。遗传程序设计是一种可变长度的表示，但它很少被用于进化其他任何东西，而不仅仅是单一的解决方案，也就是执行所有指令的单一程序。8相反，团队合作比喻将项目组织成一个单一的“组”，但可能需要在个人(项目)和团队层面上(特定任务)分配健身任务，或面临关于多样性维护的问题。

相反，TPG追求一种紧急的过程来对任务进行空间分解。因此，新的拓扑由指向种群中的其他东西的后代来识别。简单的父母和复杂的孩子之间的竞争是常态。计算总是从根节点开始，并且仅限于与该节点显式关联的程序。在每个节点评估中只能有一个“获胜”弧线，环路检测确保在检测到之前访问过的节点时必须采取不同的路径。所有这些属性都有助于确保图结构是紧急的，执行效率高，任务分解明确。实际上，这些结果在每个平台的本地分辨率下(Atari和VizDoom下分别为33600和76800像素)得到了证明，这意味着TPG继续识别非常有效的解决方案，即使在高维输入下。来自Atari平台的额外结果表明，当与更广泛的游戏和机器学习算法进行评估时，TPG仍然具有竞争力[12,15]。此外，TPG代理似乎也能够同时玩多个游戏头衔[13,15]。

当然，这代表了TPG框架早期发展的快照，因此存在许多未知因素。例如，在开发过程中是否也可以“自底向上”和“自顶向下”进行改进，或者记忆如何被引入到动态出现的基因/表型中?很明显，深度学习方法(专注于寻找合适的编码)适用于广泛的任务。相反，TPG学会了相对于原始高维状态空间对任务进行非常紧凑的分解。任务偏好和/或长处/弱点可能会导致这种模型构建方法的功能，这还有待观察。

第四章 强类型、群增强和深度学习特征选择、符号回归分类

4.1引言

符号分类(SC)是遗传编程(GP)的一个分支，它在任何全面的预测分析工具中都扮演着重要角色，尤其是因为它具有所谓的“白盒”特性。在这些最近的论文中[4,9,13]，开发了算法，将SC提升到与现有商业分类工具竞争的基本分类精度水平，包括引入GP辅助线性判别分析(LDA)[10]。在本章中，我们对我们的基本SC系统添加了一些重要的增强，并在一组理论问题和一个银行业问题上演示了它们的准确性改进。本文利用改进版的Platt序列最小优化算法[14](我们称之为MSMO)和群优化技术[5]，增强了GP辅助线性判别分析。我们添加了一个用户定义的打字系统，并将深度学习特征选择添加到我们的基本SC系统中。这个扩展算法(LDA++)在一组理论基准和一个真实世界的银行业问题上都与最好的商业上可用的M-Class分类技术有很强的竞争力。这个新的LDA++算法将遗传编程分类稳固地移动到商业上可用的分类工具的最高级别。

本文首先对前四种遗传规划SC算法进行了简要的说明，然后对所提出的扩展线性判别分析算法(LDA++)进行了详细的描述。详细描述了普拉特启发的MSMO算法，并解释了如何将LDA矩阵数学和群优化紧密地结合在一起。本文详细描述了用户自定义打字系统，并讨论了深度学习特征选择方法。

为了进行理论测试，在无噪声的情况下，构造了一组10个人工分类问题，使分类在理论上具有绝对准确的可能性。列出了这十个人工问题的判别公式。问题从线性到非线性的多模态变化，从25到1000个特征，这样每个分类算法都将侧重于从简单到非常困难的理解良好的问题。所有的理论问题都有5000个训练点和5000个单独的测试点。10种分类算法中每一种的样本外测试数据的得分报告在这里。

并不是说这五种遗传规划SC算法是最好的。事实上，我们知道有一个额外的增强算法M3GP[13]，我们没有时间为本研究实现。没有断言这五种KNIME分类算法是市面上最好的，只有KNIME是Lantern信用预测分析的可信组件。本研究旨在提供一个参考点，说明遗传规划符号分类相对于一套合理的商业分类算法的改进程度。

本文包括对五种新的SC算法和五种著名的商用分类算法的比较研究，以确定SC在竞争比较中目前的排名。五种遗传算法是:使用argmax的简单遗传规划(简称AMAXSC);M2GP算法[4];MDC算法[9]，线性判别分析(LDA)[10]，线性判别分析扩展了MSMO和Swarm (LDA++)。在KNIME系统[1]中有五种商用分类算法，分别是:多层感知器学习器(Multiple Layer Perceptron Learner, MLP);决策树学习器;随机森林学习器;树合奏学习者;梯度增强树学习者(GBTL)。

对于现实世界的测试，我们使用一个实际的银行数据集作为收到的贷款评分。训练数据包含337个特性和36223个条目，而测试数据包含相同的337个特性和额外的85419个条目。测试和训练数据是不同的。我们对五种新的SC算法和五种著名的商用分类算法进行了比较研究，以确定SC在这个现实世界问题的竞争比较中目前的排名。还包括世行的基准得分，由世行内部数据科学团队使用专有工具在多个月的时间内得出。

结果表明，在理论问题上，两种最佳分类算法是梯度增强决策树(GBTL)和本文的扩展线性判别分析(LDA++)。此外，在真实世界的银行问题上，三种最佳分类算法是梯度增强决策树(GBTL)、银行内部的数据科学方法和本文的扩展线性判别分析(LDA++)。

4.2比较算法

4.2.1 AMAXSC简介

用于多类分类的最简单的朴素遗传规划方法可以说是一种标准遗传规划方法，例如基线算法[6]的修改，使用argmax函数分类如下:



其中，C是种类的数量

每个gpk代表一个由standard演化而来的独立判别函数

遗传编程。argmax()函数选择值最高的类(1 ~ C)，与训练点属于第C类的贝叶斯概率有很强的相关性。除了标准的argmax()函数和稍作修改的遗传编程系统之外，不需要其他增强功能——修改为每个类演进一个公式，而不是通常的单一公式。

4.2.2 MDC简介

多层鉴别分类(MDC)算法是对简单的AMAXSC算法进行改进的一种进化算法。

∗gp1，∗gp2，……，wC0 +wC1∗gpC)

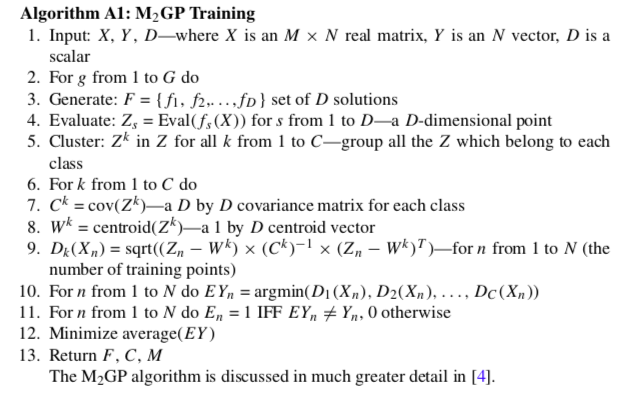
其中C为类数，gpk为GP演化公式，wij为真实权重系数(有2C权重)。

每个gpk代表一个独立的判别函数，通过标准遗传程序进化。argmax()函数选择值最高的类(1 ~ C)，与训练点属于第C类的贝叶斯概率有很强的相关性。给定一组GP演化的判别公式{gp1,gp2，…，gpC}， MDC算法的目标是优化系数权值{w10,w11,w20,w21，…，wC0,wC1}，使Eq.(4.2)对所有X和Y进行优化。

MDC算法的第一步是对每个判别条目，即wk0 + (wk1×gpk) = Yk + e，执行部分双极回归。这将产生wk0和wk1的初始权值，它们不是很好，但比random好得多。MDC算法的第二步是在选定的区分项上运行修改后的顺序最小化。这为所有判别函数产生了更好的权重候选，但仍然不是完美的。最后，MDC算法采用Bees算法[5]对系数权重进行充分优化。MDC算法在[9]中有更详细的讨论。

4.2.3 M2GP

在[4]中详细描述了M2GP算法。简单地说，M2GP算法生成了一个d维GP树而不是一个一维GP树。假设有C类，算法尝试最小化第n个训练点与第k类质心之间的马氏距离。基本的训练算法如下。



4.2.4 LDA的背景

线性判别分析(Linear Discriminant Analysis, LDA)是对Fischer的线性判别法(Linear Discriminant)的一种推广，该方法是寻找特征的线性组合，最好地分离K类训练点[2,3,12]。LDA广泛应用于统计、机器学习和模式识别。

与M2GP算法[4]之前的争论类似，我们认为任何符号回归系统都可以转换为符号分类系统。

本文从发表在[6]上的基线算法开始。我们的基线SR系统输入一个N×M的独立训练点矩阵X和一个N的相关值向量Y。SR系统输出一个预测函数，F (X) ~ Y，其中F是SR系统在分配的训练时间内可以找到的Y的最佳最小二乘估计量。F的格式很重要，它由一个或多个基函数Bfb和回归常数cb组成。总有B个基函数和B + 1的系数。下面是F的格式。



有1到B个基函数有2到B + 1个实数系数。每个基函数是对X的M个特征的运算符的代数组合，使Bfb (X)为实数。下面是一个由SR产生的预测器F (X)的典型例子。



系数c0 ~ cB在使F与Y拟合的最小二乘误差最小的过程中起重要作用。系数可以逐步演化，但大多数工业强度SR系统通过辅助健身训练技术识别最佳系数。在基线SR算法中，这种辅助健身训练技术是简单线性回归(B = 1)或多元线性回归(B > 1)。

在符号分类问题中，独立训练点X的N×M矩阵是不变的。然而，相依值的N向量只包含1到k之间的分类无序值。此外，最小二乘误差适应度(LSE)被分类误差百分比(CEP)适应度所取代。因此，我们不能在新的SC系统中使用回归来进行辅助健身训练。相反，我们可以在我们新的SC系统中使用LDA作为辅助健身训练技术。

我们的新SC系统现在输出的不是一个预测函数，而是K个预测函数(每个类一个)。这些函数被称为判别式，Dk(X)∼Yk，每个类都有一个判别式函数。SC的判别函数输出的格式总是如下所示。



argmax函数返回最大值判别函数的类索引。例如，如果Di = max(D1, D2，…，那么i = argmax(D1,D2，…，DK)。

LDA的一个核心方面是，每个判别函数都是其他判别函数的线性变化，让人想起由SR系统输出的多个基函数估计器。例如，如果GP符号分类系统产生一个具有B基函数的候选函数，那么每个判别函数具有以下格式。

图片包含 文本

描述已自动生成

4.2.5 LDA矩阵形式

如果X[n]属于类k的概率P (k|X[n])高于其他所有类的概率，则通过分配一个训练点X[n]给类k，利用贝叶斯规则来最小化分类错误率(CEP)，如下所示:



CEP的计算方法如下:



因此，公式中每个判别函数Dk都是类隶属度的贝叶斯估计百分比概率。

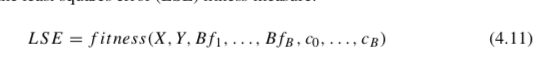


LDA的技术做了三个假设，(a)每个类都有一个多元正态分布，(b)所有类协方差相等，(c)类协方差矩阵是非奇异的。一旦做出这些假设，则最佳贝叶斯判别函数的数学公式为:



4.2.6 LDA辅助健身实施

基线SR系统[6]试图在一次运行中获得数千到数百万个回归候选项。通过返回最小二乘误差(LSE)的适应度度量的适应度函数来进行评分。



系数c0，…， cB可以作为，并返回简单的LSE。然而，大多数工业强度SR系统使用回归作为辅助适应度技术，在返回LSE适应度度量之前为系数提供最佳值。这大大提高了准确性，并允许SR将其所有算法资源集中在一组最优基函数Bf1，…，BfB的进化上。

转换为基准符号分类系统需要返回分类误差率(CEP)适应度度量，该度量定义为错误分类的计数除以Y的大小，并扩展系数以允许进行线性判别分析，如下所示:



当然系数c00，…， cKB可按原样取，简单CEP返回。然而，我们新的基线SC系统将使用LDA作为辅助适应度技术，在返回CEP适应度测量值之前为系数提供最佳值。这大大提高了准确性，并允许SC将其所有算法资源类似地集中在一组最优基函数Bf1的进化上，……, BfB。

4.2.6.4解决LDA系数的问题

通过一系列重要的先验假设，使LDA矩阵形式易于计算。这些假设包括训练数据的分布是高斯分布，所有协方差矩阵是等价的，所有类的协方差矩阵不是奇异的。不幸的是，在LDA培训过程中，这些理想的假设并不总是有效的，特别是在SC运行中，有成千上万个不同的非线性GP判别候选LDA培训尝试。当这些假设无效时，得到的LDA系数就不是完全准确的。

为了解决使用LDA系数的问题，LDA++算法将LDA矩阵的论证从统计论证改为启发式论证，在这种情况下只期望得到近似准确的结果。

由于LDA启发式不再期望获得最优结果，我们必须在LDA之后添加优化步骤，以纠正可能产生的任何错误。第二步(LDA后)是一组启发式矩阵调整，旨在纠正任何奇异协方差矩阵条件。第三步增加了改进的顺序最小优化(MSMO)，第四步增加了由蜜蜂算法[5]组成的群体智能层。

在不愉快的情况下，(4.20)和(4.21)中的协方差矩阵Ck是奇异的矩阵反转函数，(Ck)−1将失败，误差除以零。我们的启发式改变了协方差矩阵反转函数的计算机代码，这样，如果检测到一个除以零的错误，协方差矩阵Ck的对角线乘以标量1.0001。在绝大多数情况下，将协方差矩阵对角乘以这个标量会使矩阵是非奇异的。对角线调整后，重新尝试矩阵反演。当然，得到的LDA系数是不准确的，但大致是准确的。对角线调整后，如果仍然遇到除零误差，那么调整后的计算机代码将除以标量0.0001而不是零。同样的，得到的LDA系数当然是不准确的，但是大致是准确的。

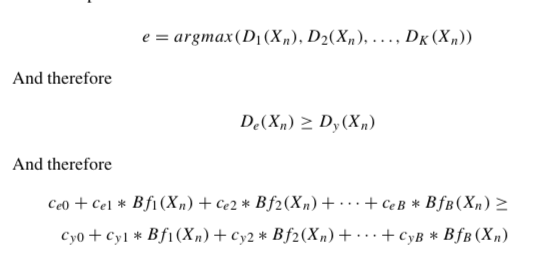
4.2.6.5改进的MSMO (Sequential Minimal Optimization)算法

LDA++算法的第三个启发式层是对Platt序列最小化优化算法的机会性修改，该算法常用于训练支持向量机[14]。在MSMO启发式的开始，对LDA系数计算训练数据集中的误分类错误百分比(CEP)。如果CEP大于10%，则跳过MSMO启发式的理论，即不值得在启发式上浪费资源，因为启发式最多可提高CEP的精度2%或3%。因此，MSMO资源只用于更好的候选人。更好的候选者有更多的进化活动。较差的候选者的进化活动更少。

在改进序贯最小优化(MSMO)层开始时，候选算法包含一个由单一系数常数集组成的群池，该群池由LDA启发式的奇异矩阵修正产生。此外，集群池中这个单条目的LDA系数的CEP也是可用的。

对于我们的MSMO算法的每次重复，在当前候选的群池中选择最适合的系数条目。如果最佳系数条目的CEP为0%，则MSMO算法将终止，就像所有分配的进化迭代都已耗尽时一样。如果CEP大于0%，则随机选择一个错误的训练点n。

自从选择训练点,n,是错误,我们知道它是估计因变量,e,不会匹配实际的因变量,y(即e̸= y)。如果K是类的数量,那么0≤e < K和0≤y < K .因为LDA使用argmax函数选择贝叶斯概率最高的判别函数,我们知道两个判别公式有以下关系。



图片包含 文本

描述已自动生成

由于K个判别式中的B基函数都是相同的，我们唯一的选择就是修改系数。使用Eq.(4.26)，我们选择基函数Bfm(Xn)，使得绝对差abs((cem∗Bf m(Xn))−(cym∗Bfm(Xn))大于B基函数中的所有其他可能选择。然后我们对系数cem和cym进行最低限度的随机改变，使Eq.(4.27)为真。

在这一点上，在MSMO算法中，使用修改后的系数来评分候选获得一个新的CEP。将修改后的系数及其相关的新CEP插入候选的群池中，并按最合适的CEP排序。然后在群池中选择最合适的系数条目，MSMO算法不断重复，直到分配的进化迭代都用尽为止。

4.2.7优化

第四步增加了由蜜蜂算法[5]组成的群体智能层。蜜蜂算法太复杂，本文无法描述。综上所述，蜜蜂算法使用一种定义明确的群体智能方法，一次对所有系数进行适应度驱动的进化修正。更多细节可以在[5]中找到。

在蜜蜂启发式的开始，训练数据集中的误分类错误百分比(CEP)被计算为最适合的系数。如果CEP大于5%，则跳过蜜蜂启发式的理论，即浪费资源在一个启发式上是不值得的，因为它最多可以提高CEP的精度0.5%或1%。因此，蜜蜂的资源只用于更好的候选者。更好的候选者有更多的进化活动。较差的候选者的进化活动更少。

4.3自定义类型系统

来自符号分类系统的无约束表达式(模型)经常被决策者拒之门外，而不考虑其有利的适应度分数。决策者通常要求SC模型具有直观的意义，并且在问题领域内具有逻辑上的可辩护性。对于高影响决策的供应链模型尤其如此。一般来说，信任SC模型的影响越大，决策者就越需要理解和相信该模型。

SC系统可以用基于模板的逻辑进行修改，以约束生成模型的演化，使其符合决策者的规范[6-8]。模板约束SC系统产生的模型很容易被决策者接受，因为这些模型总是符合决策者提供的模板。不幸的是，这样的模板约束系统不足以适应用户强类型规则。

强类型规则在不应该混合单元类型的SC应用中很重要。例如，在银行系统中，决策者可能想要禁止所有将“个人收入”与“违约数量”相加，或将“信用评分”与“种族”相乘的SC模型。如果SC模型对正确的类型感到困惑，那么决策者就会立即拒绝使用它们，即使它们非常准确，也可能会导致对SC系统整体的不信任。

4.3.1有约束的自定义模板

给定任意选择的最大深度K，构造最大二叉树固定约束模板UK是一个简单的过程，它可以覆盖在GP系统上，而不会违反所选择的最大深度极限K，也不会限制可以产生的s表达式。只要提醒我们每个f节点代表一个函数节点，每个t节点代表一个终端节点(可以是一个特征，也可以是一个实数常数)，构建模板UK的模板构造算法既简单又递归，如下所示。

文本, 信件

描述已自动生成

对于任意固定模板UK，每个f-node代表一个Lisp函数调用，如cos、sin、+、/、ln，每个t-node代表一个终端特征或常数，如x0、34.56、x10、−45.1。基本GP符号回归系统[6]包含一组函数F和一组终结点t，我们设t∈t，其中t = x0，…，xM，…IEEE实数……我们设f∈f ={+，−，\*，/，sin, cos, tan, square，等等}。现在，英国几乎成了SC中不规则s表达式的另一个模板。之所以说“几乎”，是因为英国是固定的、有规则的，而SC的s表达式是不规则的。为使模板完整，必须在F = F∪下加上一个特殊的noop函数ξ。特殊ξ函数使UK可以表示不规则的s表达式来定义ξ(a, b，…)ξ(a) = a。现在由基本GP系统产生的任何基函数都可以用UK的至少一个元素来表示。加上ξ函数，使得UK可以把由基本GP系统产生的所有可能的基函数表示到深度k。注意到，ξ函数的作用是传递函数。在英国，ξ函数允许一个固定的最大深度表达式来表示不同深度的树，例如从GP系统中可能产生的树。例如，变化的深度GP表达式



在我们的系统中，除了有ξ函数外，还提出了一种新的ξ函数。

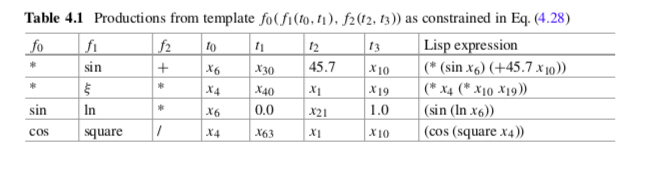
额外的细微的改变，以提高模板覆盖率，减少不必要的错误，并限制结果漫游到复数范围。所有一元函数，如cos，都被扩展为忽略任何额外参数，因此，对于所有一元函数，cos(a, b) = cos(a)。sqroot和ln函数被扩展为负参数，因此sqroot(a) = sqroot(abs(a))和ln(a) = ln(abs(a))。

在这种搜索空间的形式下，很容易计算出搜索空间的大小，很容易看出，即使是相当简单的基函数，搜索空间也是很大的。在本章中，函数集将是以下函数:F =(+−\* / abs inv cos sin tan tanh√平方立方夸脱exp lnξ)(其中inv(x) = 1.0/x)。终端是特性x0到xM−1和实际常数c，我们将其视为标准IEEE双长264。

通过显式枚举模板UK中的每个节点，以及向每个显式枚举的节点添加约束，可以将用户指定的约束添加到SC系统中。

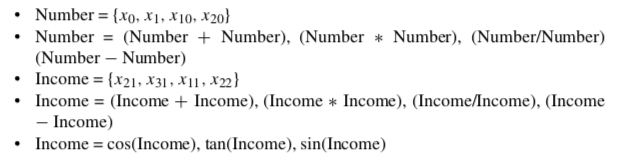
上面的SC搜索命令将生成受如下示例中显式用户声明约束的模型(记住，f0、f2、t0和t3像上面声明的那样受到约束，而f1、t1和t2则不受约束)(表4.1)。

用户约束增加了生成的SC模型被决策者接受的概率;但是，光有这些是不够的。许多应用程序还需要在SC系统中添加用户指定的强类型。



4.3.2强类型

如果能够在SC搜索命令中添加强类型规则，那么用户模板功能将得到极大的增强。这些规则可以像下面的例子那样形成。



类型规则由一个“类型”后跟一组特性或操作符类型组合组成，它们会产生指定的类型。类型规则不能像下面包含类型冲突的规则那样包含冲突，因为特性x10不能既是数字又是小部件。

4.4深度学习增强

我们使用RQL语言开发了一种深度学习策略来增强LDA++算法[7,8]。第一个深度学习层(RQL Island)通过执行一系列单独的LDA(包括SMO和Bees算法增强)来处理特性选择，使用以下RQL搜索规范在每个独立的特性上进行学习。

用上述12个非线性基函数对每个独立的特征进行同样的非线性处理。最佳排名特征(CEP)被选择为“特殊兴趣”特征组。特殊兴趣特征选择

这个最好的评分模型被输入到41个深度学习层的每一个层，其中第1层对所有20个基函数进行一般性进化搜索，所有特征都连接在一起。第2层到第21层每个层都对所有20个基本功能进行一般性进化搜索，其中只有“特殊兴趣”的功能连接。并且，每一层只发展一个基函数，保持所有其他基函数不变。从第22层到第41层，每个层都对所有20个基函数进行一般性进化搜索，并将所有特征连接起来。并且，每一层只发展一个基函数，保持所有其他基函数不变。

由于这41个深度学习层中的每一个都是紧密联系的，当其中任何一个层发现一个模型，这是一个全局适应性改进，所有处理停止，新的模型反馈到所有41个深度学习层(因此术语模型循环)。例如，经过一些进化努力，第n层发现了一个模型，这是一个全局适应度改进，例如:lda(Bg1, Bg2，…Bg20)。这个新的改进的全局拟合模型被反复馈入41个深度学习层，整个强关联模型循环过程重新开始，直到发现另一个更好的全局拟合模型。

为了使模型保持“白盒”，我们任意设置基函数的最大个数为20，任意设置每个基函数的最大深度为3(即23 = 8个终端节点)。如果精度不需要基函数，算法将其系数设置为0.0，每个基函数将演变为最优形状(受限于我们的任意最大深度3)。显然，这些参数可以针对特定的问题域进行调整。

4.5人工测试问题

在无噪声的情况下，构建了一组10个人工分类问题，以比较五种提出的SC算法和五种著名的商业有用的分类算法，以确定SC在竞争比较中目前的排名。测试问题在[11]中描述，为了方便，在附录中复制。

五种遗传算法是:使用argmax的简单遗传规划(简称AMAXSC);M2GP算法[4];MDC算法[9]，线性判别分析(LDA)[10]，线性判别分析扩展了MSMO和Swarm (LDA++)。在KNIME系统[1]中有五种商用分类算法，分别是:多层感知器学习器(Multiple Layer Perceptron Learner, MLP);决策树学习器;随机森林学习器;树合奏学习者;梯度增强树学习者(GBTL)。

每个人工测试问题都是围绕一个X训练矩阵创建的，该矩阵中填充了[−10.0，+10.0]范围内的随机实数。每个测试题的行数和列数根据问题的难度从5000×25到5000×1000不等。类的数量从Y = 0,1到Y = 0,1,2,3,4不等，取决于问题的难度。考试题目的设计从极容易到极难各不相同。第一个测试问题是线性可分的，在25列上有2个类。第10个测试问题是1000列5类的非线性多模态问题。

采用样本外测试的标准统计最佳实践。首先，训练矩阵X被[−10.0，+10.0]范围内的随机实数填充，Y类的值从下面指定的argmax函数计算。冠军是根据训练数据进行训练的。接下来，测试矩阵X填充[−10.0，+10.0]范围内的随机实数，Y类值从下面指定的argmax函数计算。之前训练的冠军将根据测试数据运行，并根据Y值得分。表4.2的结果中只显示了样本外测试分数。

所有四种SC算法(AMAXSC, M2GP, MDC, LDA)的符号分类系统使用以下运算符:

文本

描述已自动生成

4.6现实世界的银行业问题

对于现实世界的测试，我们使用一个实际的银行数据集作为Lantern Credit收到的贷款评分。训练数据包含337个特性，有36223行，而测试数据包含相同的337个特性，有额外的85419行。训练和测试数据是截然不同的，训练和测试数据是完全匿名的。337个独立特性包含分类数据和连续数据。但不包含任何FICO或类似机构的排名分数。因变量是1表示“良好”，0表示“不良”。

在领域专家的帮助下，构建了一个包含数千条强类型规则的用户定义的广泛类型文件。

目标是开发一个模型来对传入的337特性贷款申请进行评分。该模型必须通过银行内部的合规管理，并被银行内部的贷款管理团队接受。

4.7理论问题的表现

在这里，我们比较了五种提出的SC算法和五种著名的商用分类算法的样本外CEP测试分数，以确定SC在竞争比较中的排名。五种遗传算法是:使用argmax的简单遗传规划(简称AMAXSC);M2GP算法[4];MDC算法[9]，线性判别分析(LDA)[10]，线性判别分析扩展了SMO和Swarm (LDA++)。在KNIME系统[1]中有五种商用分类算法，分别是:多层感知器学习器(Multiple Layer Perceptron Learner, MLP);决策树学习器;随机森林学习器;树合奏学习者;梯度增强树学习者(GBTL)。下表按10个理论测试问题的平均CEP分数降序列出了每种分类算法。CEP越低，分类结果越准确。

积极的是，这四种新提出的符号分类算法相对于简单的AMAXSC算法有了实质性的改进，而新提出的LDA++ SC算法与最佳性能的算法极具竞争力。总体上表现最好的是梯度增强树学习者(GBTL)。倒数第二个执行器是新提出的LDA++ SC算法。

值得注意的是，这四种新提出的SC算法总体上比多层感知器学习器(MLP)性能更好。在四种新提出的SC算法中，LDA++算法的整体表现最好，并且与商用的梯度增强树学习器(GBTL)极具竞争力。事实上，除了三个测试问题外，LDA++算法在所有问题上都比GBTL做得更好。

4.8结论

已有多篇论文针对多类分类问题提出了GP符号分类算法[4,9,10,13]。将这些新提出的SC算法与五种商用分类算法的性能进行比较，表明该算法取得了一定的进展。所有四种新提出的SC算法的总体性能都优于多层感知器学习算法(MLP)。在四种新提出的SC算法中，LDA++算法是整体表现最好的，有很大的优势，并且与商业上可用的梯度增强树学习器(GBTL)极具竞争力。事实上，除了三个理论测试问题外，LDA++算法在所有问题上都比GBTL做得更好。

显然，具有商业竞争力的SC算法的开发已经取得了进展。我们现在有了一个与GBTL有很大竞争的SC分类算法。但是，在SC能够完全超越梯度增强树学习(GBTL)之前，还需要做大量的工作。我们必须对范围更广的理论问题进行比较测试，也必须对范围更广的现实世界工业问题进行比较测试。

第五章 符号回归空间的聚类分析

5.1介绍

如用于符号回归的遗传规划(GP)，往往需要处理一个非常大的数学表达式搜索空间，而这个搜索空间只会随着输入变量的数量呈指数增长。

遗传规划通过选择引导搜索，通过交叉和变异的作用发现新的模型结构。种群多样性在这个过程中扮演着重要的角色，因为它影响了算法将现有的构建块组装成越来越适合的解决方案的能力。在基因型和表型水平上的多样性之间的关系之前已经被探索过[1]，导致了一些重要的见解:

•对结构的强烈利用几乎发生在所有的运行中

•结构层面的多样性很快就消失了

•鼓励不同程度的多样性可以带来更好的表现

综上所述，我们的目标是考察GP通过语法枚举生成的符号回归模型聚类获得的邻域结构的叠加来探索搜索空间不同区域的能力。

在这篇文章中，我们集中讨论了在符号回归搜索空间中模型的分布。我们做这项工作的动机是，我们希望能够通过在搜索空间中预先计算所有符号回归模型的聚类来减少寻找拟合良好的符号回归模型所需的计算工作量。特别是，我们的目标是为符号回归模型预计算一个独立于具体数据集的相似性网络或(等效的)层次聚类。

我们在这项工作中的研究问题是:

1. 在符号回归搜索空间中模型的分布是什么?我们感兴趣的是识别相似模型的集群，其相似性可以根据模型输出(表型)或进化表达的相似性(基因型)来确定。

2. 遗传规划访问的解决方案的分布是什么?这里我们感兴趣的是GP的系统搜索偏差。特别是，(1)搜索空间中是否存在GP完全忽略的区域，以及(2)GP群体中的个体在搜索空间中的分布情况，以及从GP运行开始到结束，这种分布情况如何变化。

我们对符号回归建模目标的假设是indexregression，

下面是象征性的。我们使用这些假设作为我们研究符号回归解法的指南。

•符号回归建模的目的主要是找到开放解释的紧凑表达式。

•浅表达式优于深度嵌套表达式。

•用于现实世界回归问题的模型通常由多个项组成，这些项捕捉独立的效应。独立的项可以一个接一个地建立模型。

•现实世界的回归问题往往需要捕获模型中的非线性相关性。非线性效应往往是由一个自变量驱动的。

•输入变量和目标变量的测量是有噪声的。

5.2方法

在我们为上述研究问题寻找答案的旅程中，我们列举了完整的搜索空间(见第5.2.1节)，并计算固定输入数据的所有表达式。为了限制搜索空间的大小，我们只考虑一元函数并限制表达式的最大长度。此外，我们使用了一种语法，它限制了表达式的复杂性，并且不允许使用数值形参(即随机常数)。这进一步减少了搜索空间。

为了分析符号回归搜索空间(RQ1)中表达式的分布，我们的想法是创建所有表达式的可视化映射，希望能够让我们识别出更大的类似表达式集群。因此，我们使用所有求值表达式的集合，识别表型不同的表达式集，并确定每个表达式的表型最近邻居(见5.2.3节)。这允许我们将表达式映射到二维空间，同时保留高维空间的局部邻域结构。最近邻居的图也允许我们创建所有表达式的聚类。由于我们对两种表达都感兴趣——表现型相似的表达图谱和基因型相似的表达图谱——我们基于基因型相似性的度量创建了一个相似的图谱(见第5.2.2节)。理想情况下，我们期望在两个层次上看到相似的集群结构，假设基因相似的表达也应该是表型相似的，反之亦然

对于GP (RQ2)的搜索偏倚分析，我们的想法是重用我们在第一阶段生成的地图和集群，并分析GP是否探索了完整的地图和所有的集群。我们的想法是为GP访问的每个候选解决方案在预先计算的映射图中找到表型最相似的表达。为此，我们确定了每一代GP预先计算的集群的访问频率。我们期望GP在开始时访问许多不同的集群，在运行结束时用合适的表达式收敛到集群中。

图示

描述已自动生成

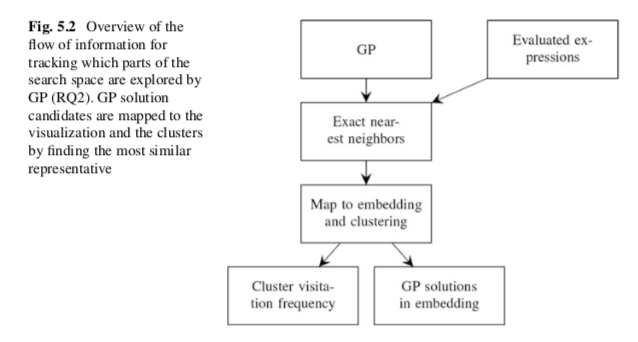
图5.1显示了搜索空间可视化和聚类的信息流概况。图5.2显示了我们如何将GP候选解决方案映射到预先计算的搜索空间和集群的映射。

我们的方法论方法的一个主要挑战是搜索空间的绝对规模。我们使用了最先进的维数约简和聚类算法来应对这一挑战，这些算法仍然适用于具有数百万个观测值和数百个特征的数据集(见5.2.4节)。这些算法的核心思想是使用随机投影树[3]来近似最近的邻居。

5.2.1语法枚举

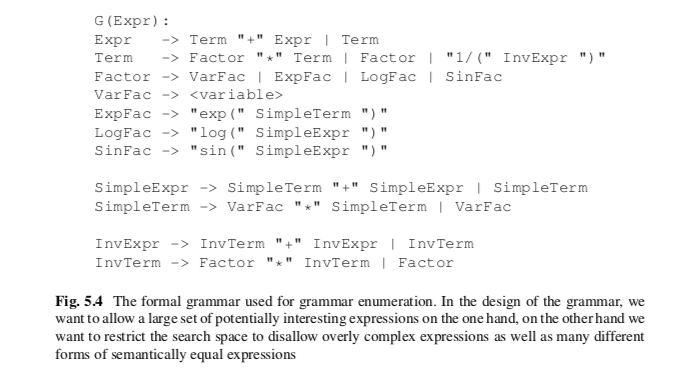
我们通过从表达式的形式语法中派生句子来创建符号回归模型，如图5.3所示。通过定义最大句子长度和省略数字常量作为开始，可以为一个问题生成一个包含所有可能模型的大而有限的集合——搜索空间。这些没有实际常数值的句子可以看作是实际模型的一般结构[5,14]。

给定一个形式语法，许多数学身份和其他表型相同但基因不同的表达式就产生了。例如，这包括交换运算符中不同顺序的参数或二项式恒等式的不同表示。为了保持搜索空间的可管理性，我们希望避免语义重复。尽管在如此大的搜索空间中完全防止所有语义重复在计算上是不可行的，但它们的数量可以在两个简单步骤中大量减少:首先，语法是受限制的，因此只能导出相关数学身份的一种表示。第二，在语法中无法阻止的标识是通过哈希来标识的:语义副本应该具有相同的哈希值。



文本, 信件

描述已自动生成



从受限语法推导出的表达式(图5.4)是项的和，它同样包含变量和一元函数，例如正弦函数或反函数。后者每学期只能出现一次。此外，函数参数的结构也受到单独的限制。

使用上下文无关的语法，无法防止语义重复，如不同顺序的术语。因此，我们另外使用一个语义哈希函数来识别语义等价的表达式。对于每个派生的符号表达式，我们用符号计算哈希值，而不计算表达式的值。通过比较先前派生的句子的哈希值，可以识别语义重复。在匹配的情况下，派生的句子很可能是语义重复的，因此被丢弃。

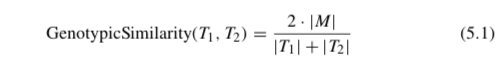
哈希函数从语法树递归地计算哈希值。树中的每个终端符号都被分配给一个常量哈希值。为了讨论交换性，像乘法或加法这样的二元运算符被平铺为n元运算符，并且它们的参数是有序的。

5.2.2表型相似性

对于表型相似性，我们使用模型输出的皮尔逊相关系数。这允许我们决定输出相似度，而不考虑函数值[4]的偏移量和规模。当我们计算表达式时，有必要假定一个有效的输入值范围。我们使用100个分布在范围(−5.0..5.0)网格上的点。所有输出向量被缩放到零均值和单位方差;未定义的输出值和无穷值被平均输出代替。这种预处理允许我们使用余弦相似度进行聚类和可视化，它被许多近似最近邻的实现所支持，相当于皮尔逊零均值向量的相关系数。

5.2.3基因型相似

我们定义了两个基于树的解决方案之间的基因型相似候选人使用Sørensen-Dice指数：



5.2.4集群化和可视化

使用表型或基因型相似性度量来可视化搜索空间的挑战之一是找到表达式到一个尽可能保留成对相似性的二维空间的映射。

我们使用t-SNE算法[7]和距离矩阵，这些距离矩阵是在前面描述的表型或基因型相似性度量的基础上计算的。

t-SNE背后的主要思想是将高维空间X映射到尽可能保留两两相似性分布的低维空间Y。数据点xi,xj∈X之间的相似性定义为xi选取xj作为其邻居的概率。一个类似的概率分布是发现在Y通过最小化Kullback-Leibler散度之间的两个分布使用梯度下降。虽然t-SNE不保留距离，但可视化可以潜在地为符号回归的搜索空间结构提供新的见解。

下面我们将更详细地描述可视化和集群化过程。

5.2.4.1基于基因型相似性的聚类与可视化

通过语法枚举得到的唯一表达式的基集约为1.6·105，对于计算完全相似矩阵来说是不可行的。因此，我们根据Keijzer-4函数[4]的R2值对表达式进行过滤，进一步将该集合缩减为一个可行的量。图5.5显示了R2值在所有唯一表达式的完整集合中的分布情况。对于基因型作图，我们只取R2 > 0.2的表达式。

我们使用HDBSCAN算法[2]进行聚类。我们尝试了两种方法:(1)基于预先计算的相似度矩阵直接聚类，(2)在映射的二维空间聚类。这两种方法产生了相似的结果。

5.2.4.2基于表型相似性的聚类与可视化

对于基于表型相似性的映射，我们决定使用完整的独特表达式集，并应用largvis实现[11]来生成可视化。LargeVis依靠近似最近邻算法实现大规模和高维数据集的可视化。在我们的分析中，我们对largivis使用了R库。LargeVis实现了t-SNE的一个变体，在这个变体中，精确的最近邻居的确定被近似的最近邻居列表所取代。这在数据点的数量上只有线性的运行时复杂性。因此，聚类和嵌入的渐近运行时间在数据点的数量上成为线性的。

该算法分为三个步骤。首先，利用随机投影树[3]确定每个数据点的近似近邻列表。第二步，计算一个稀疏加权边缘矩阵，该矩阵对最近邻图进行编码。最后，利用近似最近邻列表和边缘矩阵对t-SNE进行求解。LargeVis提供了HDBSCAN算法的一个变种[8,9]，它使用近似最近邻列表。

图表, 直方图

描述已自动生成

5.2.5映射GP方案候选方案

基于表型聚类的结果，我们研究了GP如何探索搜索空间。为此，我们在对种群中的所有个体进行评估后，在GP算法中添加一个步骤。在这一步中，我们为每个已评估的候选解决方案确定枚举搜索空间中最接近的表达式或表达式簇。这样，我们就能够计算出枚举搜索空间的访问密度。

我们预计，在进化的早期阶段，GP将探索搜索空间的许多不同领域，随着时间的推移，GP将收敛到搜索空间中包含与目标函数最相似表达式的区域。

5.3结果

在接下来的章节中，我们首先呈现基于表型相似性的聚类和可视化结果，并与基于基因型相似性的聚类和可视化结果进行比较。然后给出了对五个单变量函数的聚类性质的分析结果。最后，我们给出了GP探访频率分析的结果。

5.3.1表型映射

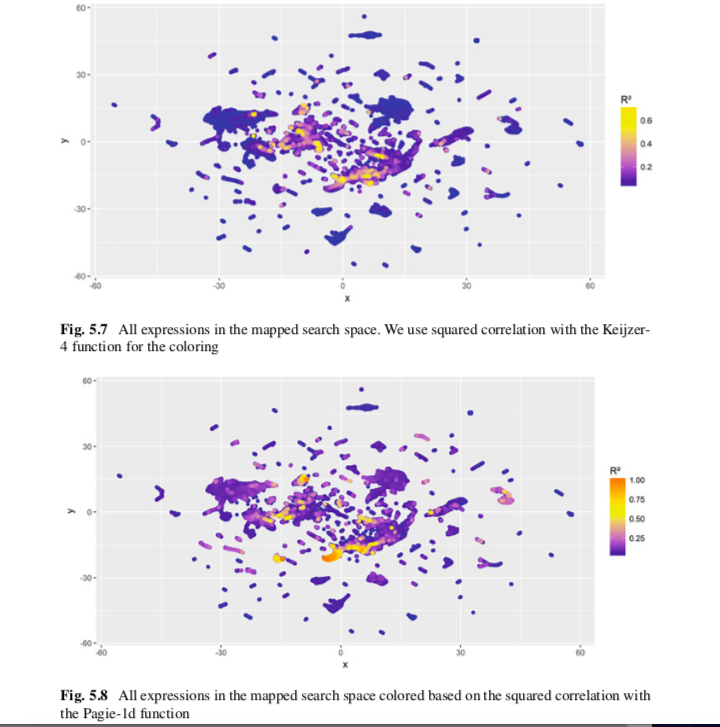
图5.6显示了基于表型相似性的可视化和聚类结果，其中我们在所有输出向量上直接使用largvis，并使用余弦相似性。每个点代表一个表达式。颜色表示一个表达式被分配给了哪个簇。可视化结果清楚地表明，在搜索空间中可以识别出几个相似表达式的聚类。

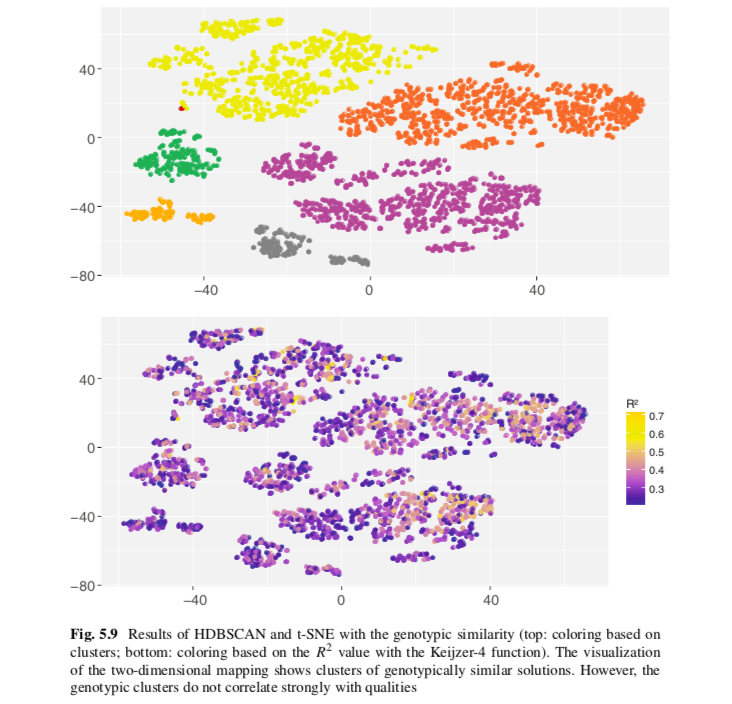
作为后处理步骤，我们为所有簇准备了图，图中显示簇内所有表达式的输出。我们发现搜索空间包括许多相当复杂的函数，并且识别出几个有趣的和相似的函数簇。图5.6为选定的部分图以及聚类中心在映射搜索空间中的位置。应该注意的是，可视化并不显示唯一的表达式，已经被HDBSCAN识别为异常值。

图5.7再次显示了表型图谱。与图5.6不同的是，这里显示了所有的表达式，我们使用了基于表达式输出与Keijzer-4函数(平方相关R2)相似性的着色方案。该可视化清楚地表明，搜索空间中只有某些区域包含与目标函数类似的表达式。值得注意的是，有几个区域包含具有较大R2值的表达式。对此至少有两种可能的解释。首先，它可能是近似t-SNE的人工产物。另一个原因可能是我们使用了余弦相似度的嵌入和R2值的着色方案。使用余弦相似度度量，两个负相关的向量是不相似的。

图5.8显示了一个不同的函数(page -1d)的相同的可视化。与图5.7相比，搜索空间的其他区域被突出显示。

结果产生的表型定位是激励进一步的研究。至少对于这两个考虑的例子，我们应该能够在映射搜索空间上使用爬山算法来找到拟合良好的表达式。搜索空间的映射必须预先计算一次，并且可以对任何目标函数重用。





5.3.2基因型的映射

图5.9显示了使用基因型相似性时t-SNE进行降维和HDBSCAN进行聚类的结果(见5.2.3节)。我们在t-SNE映射点上使用HDBSCAN。与搜索空间的表型作图相比，基因型作图并没有产生如此明确定义的集群。特别是在图5.9的下分图中，与质量值没有明显的相关性。

5.4讨论

到目前为止，我们的分析有一些局限性，值得进行更详细的讨论。

•我们只研究了单变量模型。

•语法非常有限。

•即使有7个变量引用的限制，计算工作也是

•表型聚类取决于所使用的输入值的范围

评估模型时。

•我们完全忽略了模型中数值常量的影响

在这项工作中，我们考虑了一个大约160,000个语义不同的表达式的搜索空间。然而，在图5.11中，我们看到搜索空间并不包含所有考虑的目标函数的拟合表达式。为了更深入的分析，甚至对于单变量问题，搜索空间的大小也应该加以扩展。

我们还发现，我们使用的语法产生了许多不连续函数(例如，由于除零或exp(x)的极端参数)。如果我们假设我们对这些函数不感兴趣，那么通过删除导致不连续函数的表达式，搜索空间可能会大量减少。

如果我们将分析扩展到包括多变量模型，即使我们使用相同大小的限制，搜索空间的大小也会显著增加。这只是一个结果，因为有更多不同的模型可以表达。根据初步实验，即使有两个或三个独立变量，也可以用我们上面使用的相同大小的限制来枚举搜索空间。对于三个以上的变量，有必要使用更小的大小限制。

我们还需要考虑，随着维度的增加，函数输出的多样性会变得更大，这可能导致更大的集群集合。

我们假设，对于实际问题，通常能够表示两个或三个变量的相互作用，因为可分离项可以独立建模。然而，我们不能期望这是普遍的。

考虑到语法的复杂性，我们有意地限制了可选选项的数量，以枚举整个搜索空间。然而，只要添加函数的参数的复杂性受到限制，就很容易添加更多的函数(例如幂函数或根函数)，就像我们对log(x)的参数有限制一样。

另一种可能有潜在价值的方法是计算函数的频率域的表型相似性。

5.5结论

这篇文章旨在分析GP在假设空间中的搜索行为，假设空间可视化为搜索空间的二维映射。该方法的思想是独立于待解决的回归问题列举离线阶段的完整搜索空间。为了做到这一点，我们必须定义一些限制，例如在限制语法假设下只考虑单变量函数。仍然巨大的搜索空间可以通过哈希过滤唯一表达式来进一步限制。

本章中提出的思想必须被视为一种新颖方法的第一个发现，因为考虑到搜索空间的限制限制了所声称的发现的普遍性。为了实现更深入和更普遍的理解，有必要将该方法扩展到多变量模型。同时，分析不同口味的GP和其他假设搜索技术的搜索行为也将是有趣的。

此外，一旦要解决的回归问题可用，大量离线生成的模型可以用来过滤GP运行的初始人口。类似于最近在人工神经网络社区所做的预先训练的神经网络，我们可以建立某种预先进化的初始种群来进行进化搜索:通过这种方式，我们可以为GP问题筛选一个具有基因多样性的模型子集。

第六章 用SignalGP探索进化特异性

6.1引言

在遗传规划理论与实践IX中，[20]探讨了在演化模块化程序中使用基于标记的命名。在这一章，我们继续探索基于标记的命名与SignalGP [14];我们研究了在进行基于标记的引用时不精确性的重要性:在调用可进化的名称时不精确性有多重要?此外，我们还讨论了在SignalGP上下文中可能扩展的基于标签命名的应用。

进化的名字里有什么呢?模块(例如，函数、子例程、数据对象等)应该如何在演进的程序中被引用?在传统的软件开发中，程序员手工标记模块，然后使用它们分配的标签引用它们。这种引用模块的技术是故意严格的，要求程序员精确地命名他们要引用的模块;不精确常常导致语法错误。要求不断发展的程序遵循传统的模块引用方法是不理想的:变异操作符必须做额外的工作来保证标签的正确性，否则变异的程序可能会使用未定义的标签，导致语法无效[20]。相反，依赖于更灵活、不那么严格的参考方案，遗传规划(GP)是否更好?

受John Holland使用“标签”[8-11]作为匹配、绑定和聚合的机制的启发，Spector等人[20-22]介绍并演示了一种基于标签的GP命名方案，其中标签用于命名和引用程序模块。标签是可演变的标签，是可变的，任何两个可能的标签之间的相似性(或不相似性)是可以量化的。标签允许不精确的引用。因为标签之间的相似度是可以计算的，所以引用标签总是可以链接到标签最相似的程序模块;而且，这确保了所有可能的标记都是有效的引用。因为标记是可变的，所以演进可以在演进的代码中递增地塑造基于标记的引用。Spector等人论证了可演化名称的价值，表明基于标签的命名方案通过允许程序更容易引用和使用程序模块[20]，支持具有复杂、模块化架构的程序的演化。

我们先前扩展了Spector等人的基于标记的命名方案，扩展了标记的应用来开发SignalGP[14]，这是一种GP技术，旨在提供对事件驱动编程范式的直接访问。在Spector等人最初的实现[22]中，标记被用作一种可演化的机制来标记并随后引用代码片段。在这个核心中，标签提供了通用的、可演化的专用性——一种可演化的方式来指定零个或多个带标记的实体。SignalGP扩展了标记的应用，使用它们来指定事件和事件处理程序(即处理事件的程序模块)之间的关系。然而，基于标记的命名的应用可以进一步扩展。例如，基于标记的命名可以用来标记和引用特定的指令、其他代理或其他虚拟硬件组件(例如寄存器、内存中的位置等)。在这个更广泛的环境中，标签仍然是具有定义明确的标签-标签相似性度量的可变标签，允许不精确的引用。引用标记的上下文可以限制它可以匹配的标记实体的有效集合。例如，在函数调用的上下文中，一个引用标记可能只与函数标记匹配，而在内存访问的上下文中，一个引用标记可能只与内存中的一个标记位置匹配。

在本章中，我们研究了在进行基于标记的引用时不精确的重要性，并且我们提出了SignalGP的可能扩展，使用基于标记的命名的更广泛的应用。在6.2节中，我们简要概述了SignalGP。在第6.3节中，我们使用一个环境协调玩具问题来研究在执行基于标签的引用时，允许不精确的不同阈值的有效性。我们比较了在匹配参考标签和被参考标签时，要求不同级别的标签相似度的适应度效果，从要求标签之间的精确匹配以获得一个成功的参考，到不限制标签相似度以获得一个成功的参考。我们发现，确实，在执行基于标记的引用时允许一些不精确是至关重要的。在第6.4节中，我们证明，当程序必须进化以忽略无关或误导性的环境信号时，要求标签匹配的最小相似阈值是重要的。除了提供对事件驱动编程范式的访问之外，SignalGP程序的组织方式非常适合几个有趣的扩展。在第6.5节中，我们推测SignalGP如何扩展以支持模块调控、多表示程序和个性的主要转变的几种可能性。

6.2 SignalGP

SignalGP定义了一种组织和解释遗传程序的方法，以提供对事件驱动编程范式的直接访问的计算进化。事件驱动编程范式是一种软件设计哲学，其中软件开发主要关注事件的处理(通常以来自其他流程的消息、传感器警报或用户动作的形式)[2,4,5]。事件由称为事件处理程序的代码段进行处理。在传统的事件驱动编程中，一些与事件相关的标识特征(例如，它的名称或类型)决定了触发处理事件的最合适的事件处理程序，程序员负责标记事件处理程序，以便它们处理适当类型的事件。支持事件驱动范式的软件开发环境通常抽象出事件监视和触发事件处理程序的后勤工作。这种技术简化了程序员必须在需要对来自环境或其他代理的信号做出实时反应的领域中设计和实现的代码。

SignalGP为演进的程序提供了类似的有用的抽象。在Sig- nalGP中，信号(事件)触发程序模块(函数)的执行以响应这些信号。SignalGP应用基于标记的引用技术来指定每个信号触发哪个函数，从而允许信号和函数之间的关系随着时间的推移而演变。

这里，我们在线性GP的背景下给出SignalGP的一般概述，其中程序表示为指令序列;然而，SignalGP程序的底层组织和解释是可以通过各种可进化的计算表示来泛化的(见6.5.2节)。提供图6.1以直观地指导我们对SignalGP的讨论。在[14]中可以找到更详细的讨论。

SignalGP程序(代理)是显式模块化的，由一组函数组成，每个函数将一个标记与一个线性指令序列相关联。

文本

描述已自动生成

SignalGP明确了事件的概念。每个事件都与一个标记(指示事件类型)以及其他特定于事件的数据相关联。在我们的工作中，我们将标签表示为固定长度的位串，其中标签相似性被量化为两个标签之间匹配位的比例(简单匹配系数)。由于事件和函数都被标记了，SignalGP使用基于标记的引用来确定处理事件的最合适函数:事件触发使用最匹配标记的函数，只要它在一个固定的阈值内。当事件触发函数时，函数将以事件相关的数据作为输入运行。通过这种方式，函数充当事件处理程序，基于标记的引用用作一种可演变的机制，以确定响应事件时触发的最合适的函数。SignalGP代理通过并行处理来同时处理许多事件。事件可以由环境或其他代理在内部生成，这使得SignalGP特别适合于需要程序快速响应其环境或其他代理的领域。

底层指令集被设计成允许程序容易地触发内部事件、广播外部事件以及在基于标记的上下文中工作。在SignalGP的实现中，指令是基于参数的，并且在传统的线性GP表示中，参数修改指令的效果，通常指定内存位置或固定值。除了可演化的参数，每个指令都有一个可演化的标记，它也可以修改指令的效果。例如，引用函数的指令使用基于标记的引用，当一个指令生成一个事件(例如，要在内部使用或广播给其他代理)时，指令的标记被用作事件的标记。本研究中使用的SignalGP指令集记录在我们的补充材料中，可以通过GitHub上的https://github.com/ amlalejini/ gptp -2018- exploring-evolvable - specific-with-signalgp[13]访问。

在Spector等人的基于标记引用的原始概念中，只要一个程序至少有一个标记模块，所有的参考标记都可以成功地引用[22]。SignalGP使用的基于标记的引用，无论如何，可以配置为只匹配相似度超过阈值的标记，允许程序通过避免使用类似标记来忽略事件。这个相似性阈值允许我们调整基于标签的引用成功所需的精确程度。

6.3演化名的不精确定

在调用一个可进化的名称时，不精确有多重要?基于标记的引用具有内在的灵活性，不需要标记精确匹配才能成功地引用彼此。在Spector等人最初的基于标签引用[22]的实现中，引用标签总是匹配最相似的受体标签。Spector等人推测，基于标记的引用之所以表现良好，是因为这种不精确性:任何基于标记的引用都能够找到一个只要存在的引用。然而，我们可以想象在执行基于标记的引用时允许不同程度的不精确性，从只允许相同的标记相互引用，到允许任意两个标记匹配，只要它们是最相似的一对。事实上，为了成功的引用，任何最低程度的标签相似度都可以被强加(例如，要求标签至少有50%的相似度才能被认为是最佳匹配)。

这里，我们将探讨使用SignalGP进行基于标记的引用时不精确的重要性。我们进化SignalGP代理来解决一系列相似度阈值下的环境协调问题，从0%(没有相似度要求)到100%(需要完全匹配)。

6.3.1环境变化问题

不断变化的环境问题是一个玩具问题，我们设计它是为了测试GP程序对环境信号作出适当反应的能力。我们以前使用这个问题来演示使用SignalGP[14]的事件驱动范式的价值。

变化环境问题要求agent不断将其内部状态与随机变化环境的当前状态相匹配。环境被初始化为一种随机状态，在随后的每一个时间步骤中，环境有12.5%的机会随机改变为16种可能状态中的任何一种。要想成功，代理必须监视环境的变化，并适当地调整它们的内部状态。

环境变化产生具有特定环境标签的信号(事件)，这些信号将触发适当的SignalGP功能;通过这种方式，SignalGP agent可以对环境变化做出反应。这16种环境状态中的每一种都与一个不同的标记相关联，该标记是在运行开始时随机生成的。代理通过执行16条状态改变指令(每种可能的环境状态对应一条指令)中的一条来调整其内部状态。因此，这个问题的最优解决方案是一个16函数的程序，其中每个函数由不同的环境信号触发，当触发时，函数会适当调整agent的内部状态。我们的补充材料中记录了一个解决环境变化问题的示例解决方案，可以通过GitHub上的https://github.com/ amlalejini/ gptp -2018- exploring-evolvable - specificing-with-signalgp[13]访问。

为了探究基于标记的引用的不精确性的价值，我们在9种处理下进化了30个重复的SignalGP药物群体，每个都需要不同的事件触发功能的相似阈值:0%，12.5%，25%，37.5%，50%，62.5%，75%，87.5%和100%。注意，当执行基于标签的引用时，如果最接近的匹配标签不大于或等于所需的相似度阈值，引用失败。

6.3.1.1假说

100%相似度阈值等同于精确名称引用;因此，我们预期它会表现不佳。0%的相似度阈值相当于[22]在他们最初基于标签的引用演示中使用的阈值;因此，我们期望它表现良好。然而，中间阈值是否同样有效?它们提供了不同程度的允许的不精确性，同时允许程序被动地忽略一些传入信号。在之前使用SignalGP[14]的工作中，50%相似度阈值在变化环境问题上表现良好;因此，我们期望具有中间阈值的处理比需要精确的引用标记匹配才能成功的运行表现更好。

6.3.1.2实验参数

对于每一种治疗，我们从一个简单的祖先程序(由8个无操作指令组成的单一功能)开始，在10000代中进化出30个重复的1000种药剂群体。我们用唯一的随机数种子初始化所有复制。每一代，我们在人群中评估所有药物三次(三次试验)。每次试验由256个时间步长组成，agent在一次试验中的得分等于agent内部状态与环境状态匹配的时间步长。因此，可能的分数从0到256不等。代理人的适应度是在三次试验后获得的最低分数，因此选择的代理人表现一致。我们使用精英和锦标赛(大小为4)的组合选择来确定哪些个体每一代都进行无性繁殖。

使用signalgp感知突变算子对后代进行突变。我们使用了全函数复制和删除操作符，每个函数的应用速率为0.05;这些运算符允许进化来调整SignalGP程序中函数的数量。我们以每位0.005的突变率对指令和函数标记进行突变。我们以每条指令/参数0.005的速率应用指令和参数替换。我们以每条指令0.005的速率应用了单指令插入和删除操作符;当单指令插入发生时，我们插入一个带有随机参数和随机标记的随机指令。除了单指令的插入和删除，还可以通过滑动突变操作符[15]插入或删除指令序列。当触发时，滑动突变可以复制或删除一个函数中的多指令序列。我们以每函数0.05的速率应用滑动突变。

代理被限制为最多16个总功能，每个功能被限制为最多32个指令长度。代理被限制为最多32个并行执行线程。座席被进一步限制在每个呼叫堆栈128个呼叫状态。所有标签都表示为长度为16位的字符串。

6.3.1.3数据分析

我们分析了进化过程中两个时间点的进化种群:第1000代和第10000代。对于分析的每个群体，我们提取了性能最好的程序，并对其进行了100次评估(考虑到环境随机性)，使用其平均性能作为其代表性适应度。对于分析的每个时间点(第1000代和第10,000代)，我们比较了不同处理过程中进化程序的性能。为了确定在一组治疗中是否有任何治疗具有显著性(p < 0.05)，我们进行了Kruskal-Wallis检验。对于Kruskal-Wallis检验显著的时间点，我们进行了事后配对Wilcoxon秩和检验，对多重比较应用Bonferroni校正。所有统计分析均在R 3.3.2[18]中进行。

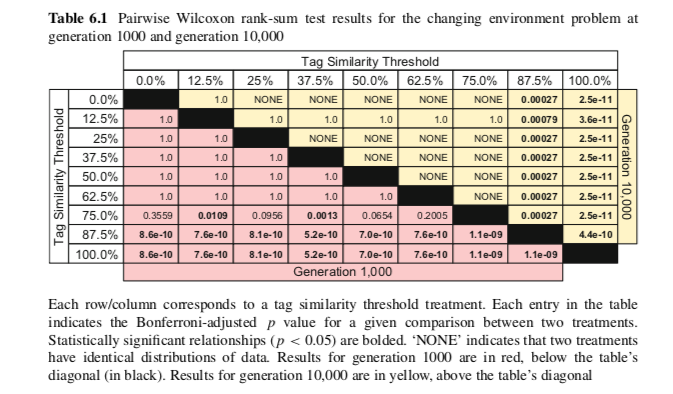
我们所有的可视化结果都是使用seaborn Python库[23]生成的。运行我们的实验、执行统计分析和生成可视化的代码可以在Github[13]上公开获取。

6.3.2结果与讨论

图6.2给出了实验早期(第1000代)和实验结束时(第10000代)环境问题变化的结果。在第1000代和第10,000代，在不同相似度阈值下演化的程序表现出显著差异(Gen. 1000: Kruskal- Wallis检验，卡方= 161.27,p < 2.2e−16;Gen. 10,000: Kruskal-Wallis检验，卡方= 221.72,p < 2.2e−16)。表6.1给出了我们在第1000代和第10,000代的结果的事后配对Wilcoxon秩和检验结果。

图形用户界面, 应用程序

描述已自动生成



在经过10,000代的进化后，87.5%和100.0%相似阈值处理所进化的程序仍然显著低于低相似阈值处理所进化的程序。然而，经过10,000代，一些在87.5%相似阈值处理下进化的复制能够产生最优程序。在100.0%相似度阈值处理(精确姓名匹配)中，没有最优的程序进化。详细的统计结果可以在我们的补充材料[13]中找到。

在调用基于标记的名称时，允许一些不精确性是至关重要的，因为我们将代理限制为最多16个总函数，优化解决16个状态变化的环境问题需要程序将其16个可能的函数中的每一个用于响应特定的环境状态。每个函数都必须进行标记，以便只有单个环境状态更改才能触发它，当触发时，该函数必须立即适当地更新代理的内部状态。如果事件触发函数需要精确的标记匹配(100%相似性阈值)，则每个函数的标记必须进化到与单个环境状态标记逐位匹配。正如预期的那样，我们的结果表明，要求标签精确匹配成功的参考文献会阻碍进化:经过10,000代，在100.0%相似性阈值处理下，没有最优程序进化出来。

在执行基于标记的引用时允许不精确的处理中，每个函数的标记必须进化到与单个环境状态紧密匹配(高于给定的相似性阈值);更高的最低要求相似度阈值需要进化来更精确地调整功能标签。正如87.5%的相似性阈值处理所证明的那样，要求极高的精确度会阻碍进化适应。

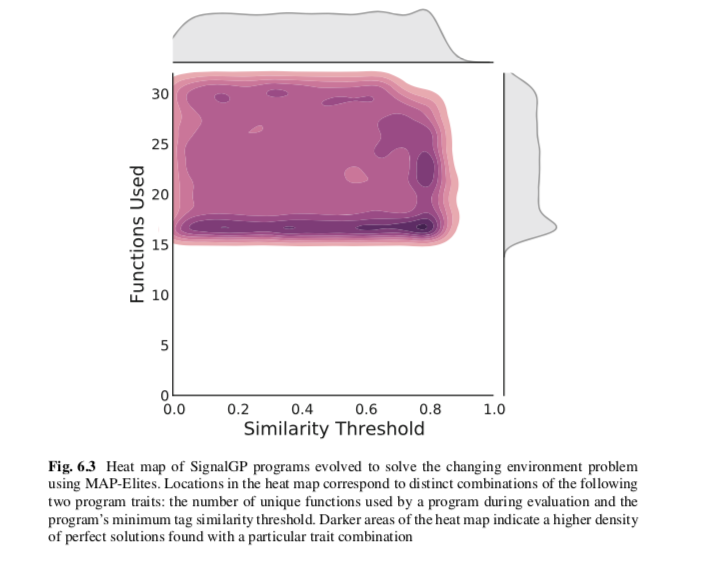
经过1万代，所有处理在相似阈值低于87.5%时，方案性能无显著差异。尽管允许基于标签的引用不精确对于进化程序以解决变化的环境问题是至关重要的，但事实证明，所需的中间精度水平(12.5%、25.0%、37.5%、50.0%、62.5%和75.0%相似度阈值)与不施加任何标签相似度约束(0.0%相似度阈值)一样有效。

6.3.2.1MAP-ELITES

我们使用MAP-ELITES[17]进化算法来进一步阐明使用基于标记的命名模式时不精确性的重要性。在地图精英中，一个群体是基于一组进化解决方案的选择特征构建起来的。每个选择的特征在网格上定义了一个轴，每个细胞代表选择的特征的不同组合;此外，每个细胞只保留最适合的(精英)解决方案，发现与细胞相关的特征组合。通过随机生成解决方案并将它们放置到网格中相应的单元中(基于随机解决方案的特征)，可以初始化地图精英网格。初始化后，被占用的细胞被随机选择复制。当一个解决方案被选择用于繁殖时，我们会产生一个突变的后代，并确定该后代在网格中的位置。如果细胞是空的，新的溶液被放置在那个细胞中;否则，我们将比较新的解决方案与当前居住者的适用性，保持两者中更合适的。随着时间的推移，这个过程产生了未来解决方案的网格，跨越了我们用来定义网格轴的特征范围。

Dolson等人扩展了地图精英的使用，以检查GP表示[3]。通过选择与程序架构相对应的地图精英网格轴，我们可以得到一个快照，了解哪些类型的程序能够成功完成任务，以及在所选的特征之间可能存在哪些权衡。我们使用这种方法来探索SignalGP中不精确性的作用:我们将地图精英算法应用于变化的环境问题，使用基于标记的引用的最小相似阈值和程序在评估期间使用的唯一函数数来定义我们的地图精英网格轴。在我们更传统的进化实验中，我们为每种处理锁定了所需的最小相似性阈值。在我们的地图精英分析中，我们允许程序的最小相似阈值在0.0%到100.0%之间演化。此外，我们将程序中允许的函数数量从16个增加到32个。

我们用1000个随机生成的SignalGP程序初始化我们的地图精英网格。我们运行了10万个世代的地图精英算法，每个世代代表1000个繁殖事件。我们运行了50个复制地图精英运行，为我们提供了50个不同的解决方案网格变化的环境问题。在每次运行结束时，我们过滤掉了在我们的50次运行中不能完美解决问题的任何程序。图6.3中的热图显示了在我们选择的性状空间内最优程序的密度(在运行中聚合)。



从图6.3可以看出，所有最优程序都使用了16个以上的函数。这并不奇怪，因为不断变化的环境问题不能用少于16个函数来最优地解决。图6.3所示的所有进化解中，相似性阈值最高为87.4657%，支持基于标记的引用过于精确会阻碍进化的观点。

6.4噪音

忽略环境中的信号有什么价值呢?在一些问题领域，完全忽略分散注意力的、不相关的或误导性的信号，同时监控其他信号的能力是至关重要的。例如，在一个拥挤的餐馆里，选择性的注意力可以让我们忽略背景噪音，只关注一个单一的转换。

在这里，我们将误导的分心信号合并到变化的环境问题中，以证明在SignalGP上下文中忽略信号的价值。在SignalGP中，基于标记的引用的0%相似性阈值防止代理被动地忽略环境中的信号(事件)。SignalGP程序仍然可以组织成通过使用适当标记的无效函数来消耗信号或基于特定事件的数据过滤信号来主动忽略信号。在SignalGP中，基于标记的引用的100%相似性阈值会导致SignalGP程序忽略任何标记与代理的功能标记不完全匹配的事件，我们已经证明这阻碍了进化(6.3节)。然而，中间相似阈值允许SignalGP代理被动地忽略环境中的信号，而不会阻碍自适应进化。我们探讨了允许不同程度的被动信号识别的价值，通过不同的相似阈值在SignalGP中使用分散的环境问题。

6.4.1 The Distracting Environment问题

The Distracting Environment问题与变化的环境问题(在第6.3.1节中描述)相同，但增加了随机发生的分心信号。与不断变化的环境问题一样，环境可以在任何时候处于16种状态之一，每次更新都有12.5%的机会进行更改。每一步都有12.5%的机会发生干扰事件，不受环境变化的影响。正如我们随机生成与16种环境状态相关联的16种不同的标签一样，我们也生成16种不同的干扰信号标签，它们保证不与环境状态标签相同。因此，为了成功，代理必须监视环境的变化(适当地调整它们的内部状态)，同时忽略误导性的干扰信号。

我们用相同的实验处理和参数重复了第6.3节中描述的实验，但是是在分散的环境问题而不是变化的环境问题的背景下。

6.4.1.1假说

就像在不断变化的环境问题中一样，在分散的环境问题中，最佳性能需要16个功能，每个功能都被标记为由单一的环境状态信号触发;一旦触发，函数必须适当地调整代理的内部状态。然而，分心的环境问题也要求代理忽略分心的信号。如果分散信号能够触发函数，代理就不能可靠地维护与当前环境状态相匹配的内部状态。考虑到agent必须有16种功能来调整内部状态以响应环境的变化，它们必须能够被动地忽略干扰信号，以避免触发错误的内部状态。因此，0%相似阈值处理不能产生性能最佳的程序。此外，中间相似性阈值必须足够高，以允许代理被动地区分分心信号和环境变化。我们期望具有较高的中间相似性阈值的处理能够达到最优。

6.4.1.2统计方法

我们分析这些数据的统计方法与第6.3.1.3节中描述的方法相同。

6.4.2结果与讨论

图6.4给出了实验早期(第1000代)和实验结束(第10000代)时分心环境问题的结果。在第1000代和第10,000代，在不同相似度阈值下演化的程序表现出显著差异(Gen. 1000: Kruskal-Wallis，卡方= 144.3,p < 2.2e−16;Gen. 10,000: Kruskal-Wallis，卡方= 193,p < 2.2e-16)。表6.2给出了我们在第1000代和第10,000代的结果的事后配对Wilcoxon秩和检验结果。

图形用户界面, 应用程序

描述已自动生成 表格

描述已自动生成

在不断变化的环境问题中，运行需要精确的名称匹配(100%标签相似性阈值处理)的程序，其性能明显低于在所有其他处理中进化的程序。在第1000代和第10,000代，75%标记相似性阈值处理的程序显著优于其他所有处理的程序。到10000年

结果表明，只有75%和87.5%的标记相似度阈值处理能得到最优的结果。详细的统计结果可以在我们的补充资料中找到，可以通过GitHub[13]获取。

在调用基于标记的名称时要求一定的精度也很重要

这些数据并不令人惊讶:我们将分散注意力的环境问题设计成一个玩具问题，以证明在使用基于标记的引用时，有时需要一定的精确度是很重要的。因为我们将程序限制为16个函数，而所有的16个函数都被要求监测环境的变化，解决分散注意力的环境问题需要程序有能力区分真实的、有意义的信号和不相关的、无意义的信号。然而，即使在这种情况下，信号识别是至关重要的，需要精确的标记匹配信号才能成功地触发程序功能，对于性能良好的程序来说，仍然是一个苛刻的要求。

对于变化的环境和分散的环境问题，75%的标签相似度阈值处理产生了最优的执行程序，允许在分散的环境问题中充分的信号辨别，同时不太严重地阻碍进化的能力，引导程序响应真实信号。然而，这些数据并不一定意味着75%的标签相似性阈值。令人分心的环境问题的重要的标签相似度阈值的数量取决于干扰信号,必须忽略与真实信号的数量项目必须回应,组成一个标签的比特数(在这里,我们使用16),以及函数SignalGP程序允许的数量。

6.5进化的名称中还有什么?SignalGP中基于标记命名的扩展应用

到目前为止，我们已经在SignalGP的上下文中探索了可进化名称中不精确的重要性。在本节中，我们将讨论对SignalGP框架的几个可能的扩展，因为其基于标签的命名方案提供了演化的特异性。

6.5.1 SignalGP功能调节

将GP和基因调控网络的观点结合在一起并不新鲜[1,16]。调节基因型表达的能力在生物和计算系统中都是有价值的，它允许环境反馈在个体的一生中改变表型性状。

SignalGP可以很容易地扩展到建模基因调控网络，其中功能可以上调(即，更有可能被标记引用)或下调(即，更不可能被标记引用)。例如，一个通常不会被事件触发的函数可以被上调，以提高其相对于其他更接近的函数的优先级。我们可以在增加或减少功能调节修饰符的指令集中增加调节指令，使用基于标记的引用来确定哪个功能应该被特定指令调节。

基因调控为表型灵活性提供了另一种机制，允许信号藻类程序在响应环境反馈时改变参照关系。这种机制可能对需要终身学习或一般行为可塑性的问题有用。

6.5.2 Multi-Representation SignalGP

在这项工作和之前的工作中，我们在线性GP的上下文中专门使用了SignalGP: SignalGP函数将标签与线性指令序列相关联。然而，原则上，SignalGP是可以在各种进化计算表示中泛化的。

SignalGP程序由一组函数组成，其中每个函数都通过它的标记来引用。我们可以把这些函数想象成黑箱输入-输出机器:当被一个事件调用或触发时，它们与输入一起运行，并通过操纵内存或生成信号来产生输出。我们在SignalGP函数中专门使用线性GP;然而，我们也可以很容易地使用其他类型的表示，能够接收输入并产生输出(例如，其他GP表示，人工神经网络，Markov大脑[6]，硬编码模块，等等)。我们甚至可以在单个代理中使用多种表示。

SignalGP基于标签的命名方案所提供的进化特异性允许我们使用这种黑盒隐喻。由不同表示组成的函数仍然可以通过标记相互引用，事件与用于处理它们的底层表示无关，只需要表示能够处理特定于事件的数据。允许这些类型的多表示代理可能会使SignalGP虚拟硬件、程序评估和变异操作复杂化，但它将为进化提供一个不同表示的工具箱。

Hintze等人提出并演示了进化自助餐方法，其中马尔科夫大脑[6]可以由异质计算基板组成，允许进化为给定的问题[7]找到最合适的表示。进一步，Hintze等人的自助餐方法证明了混合解的成功。多表示SignalGP为进化多表示代理提供了一种尚未探索的替代方法，将Buffet方法引入事件驱动的环境中。

6.5.3 SignalGP中的主要转换

在个体性的重大进化转变中，以前截然不同的个体联合起来形成一种新的、更复杂的生命形式，重新定义了个体的意义。真核生物的进化、多细胞生命和群居昆虫群体都是个体进化的例子。通常，组成较高层次实体的个人只能获得局部信息，无法直接访问较高层次单位的全局状态;下级单位必须依靠信号和感觉信息来协调它们在群体中的作用[19,24]。在计算的意义上，个性的重大转变是分布式系统的进化。在GP中捕获这些类型的转换将为进化提供一种机制，使其从以前的单个程序逐步形成分布式系统。

在前一节中，我们描述了如何扩展SignalGP，以允许函数(模块)可以是任何能够接收输入并产生输出的表示形式的多表示程序。我们可以把这种方法更进一步地应用到多表示的SignalGP上:SignalGP代理中的任何模块都可以是另一个(以前的)SignalGP代理。这种方法在概念上类似于复杂的程序图表示[12]。

我们可以想象一个突变算子，当它被应用时，通过将共同进化的SignalGP程序作为自包含的、带标签的模块注入到被突变的程序中来诱导个体的转变，允许单个个体成为较低层次个体的聚合。此外，个性的转变可以层次化地应用。生物进化中也有这样的层次转变的例子:群居昆虫群体由许多多细胞个体组成，每个多细胞个体又由许多真核细胞组成，真核细胞又由细胞器组成(许多细胞器被认为是形式上不同的个体)。

6.6结论

在这一章中，我们探讨了在GP中调用基于标记的名称时不精确的重要性。我们表明，在执行基于标记的引用时允许不精确对于快速自适应进化是至关重要的。相反地，当一些信号需要被忽略时(例如在我们分散注意力的环境中)，防止不同的标记找到不正确的匹配是至关重要的。因此，标签相似性的中间阈值可能是在广泛的环境中最优进化的理想选择。对于一个给定的问题，最合适的相似度阈值取决于问题的细节和所使用的表示方式。例如，我们需要考虑在特定问题中使用标记的方式，以及如何表示、变异和比较这些标记。

有趣的是，虽然精确命名是人类程序员最直观的引用机制，但当允许程序引用不精确时，进化就会成功得多。事实上，突变-选择平衡可能会阻止精确的参考文献在进化过程中稳定地维持。如果标记发生了突变，以至于我们期望在每次复制事件中至少有一个程序标记(引用或引用)发生突变，那么引用和引用标记之间的关系就不太可能稳定地保持。

SignalGP和我们对SignalGP的扩展都是受到生物系统和过程的启发。随着我们继续开发SignalGP，我们的目标是继续推动GP的边界，并使用SignalGP作为工具来研究启发其发展的自然系统，如模块化的进化，基因调控，细胞信号传导，以及主要的个体进化转变。

第七章 Lexicase选择算法

7.1 引言

Lexicase选择是用于进化计算系统的选择算法，用于确定哪些个体将被允许为进化过程中的子孙后代做出贡献。尽管它已用于幸存者选择[ 8 ]，但其主要用途是作为父代选择算法，选择要提供给遗传算子的个体。遗传算子，例如突变和杂交，使用选定的父母作为原材料来构造孩子。Lexicase选择通过过滤个体池来选择个体，该个体池在过滤之前通常包含整个种群。过滤是分步完成的，每个步骤根据单个测试用例（输入/输出对）的性能进行过滤。一次将测试用例视为一个随机顺序。Lexicase选择已经在遗传编程系统最广泛的测试，它已被证明优于在若干上下文其它选择方法[ 2 - 5 ，7 ，9 - 11 ]。但是，在其他情况下选择词法酶的有效性尚未得到充分探讨。在本文中，我们调查了lexicase选择在具有线性，固定长度基因组的传统遗传算法中的效用。我们选择这个框架的部分原因是，传统遗传算法的大量文献为解释我们的结果提供了广泛而深入的背景。

利用几种遗传算法求解的传统遗传算法是随机生成布尔约束满足问题[1]。虽然这些问题来源于布尔可满足性问题，但问题解决者(在这里是遗传算法)不能访问约束本身，也不能访问约束所包含的变量。相反，问题解决者只能访问一个过程，这个过程决定每个约束是否通过对所有变量的真值分配得到满足。至关重要的是，问题解决者没有得到关于哪些约束可能依赖于哪些约束的信息。这个设计的目的是使问题作为抽象的问题在许多现实世界的领域,我们可以告诉候选人解决方案是否满足约束,但是我们没有进一步的信息约束的性质,或不同的约束可能是相互依存的。

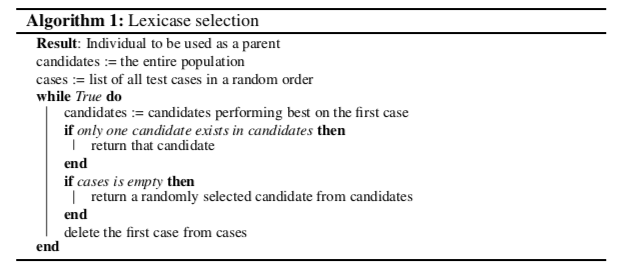
在本章中，我们给出了用传统遗传算法和字典库选择来解决布尔约束满足问题的实验结果。我们比较了使用lexicase选择算法的性能与使用更传统的选择算法运行同一算法的性能，特别是适配-比例选择和竞赛选择(具有多个竞赛规模)。

在下面几节中，我们首先描述本研究的重点lexicase选择算法。然后我们描述我们在实验中使用的布尔约束满足问题，以及我们的实验方法。然后我们展示我们的结果，并讨论它们对未来工作的影响。

7.2 Lexicase选择算法

Lexicase选择是一种从种群中选择个体作为源材料的方法，在此基础上遗传操作符，如突变和交叉，为下一代构建后代。Lexicase选择的独特之处在于，它允许选择依赖于多个评估标准及其所有的组合，而不需要将这些标准汇总为整体的“适合度”值。这与遗传规划中使用的传统选择方法不同，传统选择方法要求为每个候选解分配单个标量值，以便指导搜索。

在lexicase选择之前的大多数工作中，它已被用于遗传规划系统选择父程序，然后受到变异产生下一代程序。在这种情况下，评估标准通常是程序在不同输入上的错误，这在遗传规划文献中经常被称为“适合度案例”或“测试案例”。



在lexicase选择中，每当需要一个亲本时，最初包含整个种群的个体池在连续的阶段中进行筛选，直到剩下一个个体并被选中。在第一阶段，只有在随机选择的测试用例中表现最好的个体被保留。如果超过一个个体留下，第二个随机选择的测试用例将用于下一阶段的筛选。这个过程不断重复，直到只剩下一个个体，或者直到测试用例已经用尽，在这种情况下，从剩余池中选择一个随机个体。算法1提供了最常用的lexicase选择形式的伪代码。

有时，lexicase选择会选择性能较好的个体，而不是少量的测试用例。在许多需要将所有测试用例的性能聚集到单个标量值的选择方法下，这些“专家”中的许多人很少被选择来进行复制和变异。即使专家解决的案例中有一个是大多数人难以解决的，情况也往往是这样。其原因是选择压力均匀分布的假设，问题的所有部分都是同等困难的。“专家”被忽略了，即使他们更擅长问题的某些子集，可能包含手头任务的部分解决方案。通过允许这些“专家”为下一代做出贡献，lexicase选择允许后代可能包含问题的一个特定子集的解决方案。与出于类似动机而开发的其他方法(如“隐性适应度共享”和“确定性拥挤”)进行比较的词库选择在其他地方有介绍。

一些变异的lexicase选择方法也被开发和研究。例如，epsilon lexicase selection是一种变体，其中的候选项在当前情况下不是严格意义上的“最佳”，但“足够接近”(对于epsilon的某些定义，在epsilon中)，并被证明在浮点错误问题上特别有效。其他变体已在遗传规划理论和实践研讨会[12]的先前实例中开发和探索。然而，在目前的研究中，我们使用了上述方法中最简单和最标准的版本。

7.3问题

7.3.1布尔约束满足

本研究实验所使用的问题为布尔约束满足问题[1]，该问题是根据三个参数随机生成的:变量总数v，约束数量c，每个约束数量n。

每个子句是三个字面值的析取，每个字面值要么是一个变量，要么是一个否定变量。每个约束都是子句的连词，而问题是约束的连词。如果在赋值的上下文中所有约束值都为真，那么将真值赋给变量就是一个解决方案。

这些问题在某些方面类似于用3-CNF(连接范式，每个子句有三个字)表达的布尔可满足性问题，将子句分组形成约束。然而，与标准可满足性问题的情况不同，例如在sat求解比赛[6]中使用的那些问题，我们不允许我们的问题求解者看到公式本身，也不允许有关于哪些变量出现在哪些约束条件中的任何信息。问题解决者可以就每个约束对变量的真值赋值进行评估，确定赋值是否满足每个约束，但这是问题解决者收到的关于问题的唯一信息。

7.3.2随机问题产生

我们通过对所有变量的真值随机赋值来产生一个问题。这个分配将是生成问题的一个解决方案，但是我们将在生成问题后丢弃它，问题解决者的任务将是重新发现这个分配，或者发现另一个也满足问题中的所有约束的分配。

一旦我们有了所有变量的随机分配的真值,我们自己用一个简单的生成问题,迭代generate-and-test算法:我们创建一个指定大小的随机集的约束(其中可能包括重复的条款,可能在不同的约束),我们检查评估为true用给定的任务。如果是这样，那么我们就把约束条件作为我们实验的问题;如果没有，那么我们随机生成一组新的约束，重复这个过程，直到我们找到一个满足赋值的约束。

对于每个(v,c,n)三元组，我们产生了15个不同的问题。每种问题解决方法在这些问题上运行了50次，结果是每种问题解决方法在v、c和n的每种组合上运行了750次。结果通过对每种参数组合的750次运行的平均值来评估。生成问题所使用的具体参数，以及用每种方法对每个问题进行的运行次数如表7.1所示。

表格

描述已自动生成

7.4实验方法

7.4.1遗传算法

我们的问题解决方法都是同一遗传算法的实例，除了亲本选择方法外，参数相同(见表7.2)。

种群中的个体是真值赋值，每个变量由基因组成的基因组，表明该变量在指定赋值中是否为真或假。

我们使用人口的代际遗传算法开始随机个体,然后进入一个无限循环中,每一代,所有个人都是检测错误,父母选择的生产孩子那些错误的基础上,和孩子们产生的不同选择的父母。

7.4.2变异

在变异步骤中，选择作为亲本的个体首先(可能)进行交叉，然后(可能)进行突变。

采用标准的单点方法作为交叉算子，允许在随机选择的交叉点进行亲本重组。交叉率为0.9，这意味着90%的孩子是由交叉生育的。

对于突变，使用了位翻转突变操作符，允许染色体中随机选择的位被翻转。基因突变率为0.1，这意味着10%的儿童会发生基因突变。

7.4.3父母选择

我们比较了lexicase亲本选择、tournament亲本选择和适合度比例亲本选择。所有的选择方法都进行了替换选择;也就是说，同一个体可能在同一代中被多次选择为亲本。

在比赛选择和适应度比例选择中，根据个体满足的约束数确定个体的总误差值。如果个体满足所有约束条件，则其误差为0。否则，它的误差值是它不满足的约束的数目。

对于比赛大小为t的比赛选择，我们首先建立一个由t个个体组成的比赛集合，每一个个体都以均匀概率(有可能重复)从整个种群中被选中。然后，作为选中的父节点，我们返回锦标赛集合中总误差最小的个人。

较高的比赛规模会使比赛选择更具选择性，也就是说，总误差高的人被选中的可能性较小，而较低的比赛规模则会降低选择的选择性。因为我们的初步实验表明，选择性较低的设置似乎表现得更好，我们想要考虑比竞赛规模为2的比赛选择更少的方法，这通常被认为是最小的，因为竞赛规模为1的比赛选择等同于完全随机选择个体。为此，我们采用了一种惯例，即对于一个非整数值的竞赛规模t，在1到2之间，我们将使用竞赛规模2的概率为t−1，并完全随机地选择一个父节点。例如，在t = 1.25时，有25%的时间我们随机选择2个个体，并返回总误差较小的一个作为所选的父类;剩下的75%的时间，我们会返回，作为被选中的父节点，一个从整个总体中以均匀概率被选中的父节点。

我们以标准的方式进行适合度比例选择:满足si约束的个体i的选择概率是si除以总体中所有个体j的sj之和。在没有个体满足任何约束的退化情况下，会产生一个分母为零的个体被随机选择。

7.5结果

7.5.1亲本选择法成功率

表7.3显示了我们的主要结果，比较了遗传算法在适合度成比例的亲本选择、竞赛亲本选择(竞赛大小为2)和lexicase亲本选择时的成功率。

表7.3包含一行的数量为每个组合变量(v)和(c)的约束。所有运行的指定值v和c是聚合在相应的行,不管每个约束(n)的数量条款。因为我们进行了750 v的运行与每个组合,c, n为每个家长选择方法,因为我们对5进行了大量实验不同的n值(见表7.1),表7.3中的每一行报告来自5∗750 = 3750的数据，使用表中列出的三个父选择方法。

表7.3中报告的数字是成功率，定义为产生成功解决方案的总运行的比例(误差向量仅由0组成)。在每一种情况下，Lexicase选择产生最高的成功率，并且Lexicase选择提供的改进在大多数情况下具有统计学意义。

表格

描述已自动生成

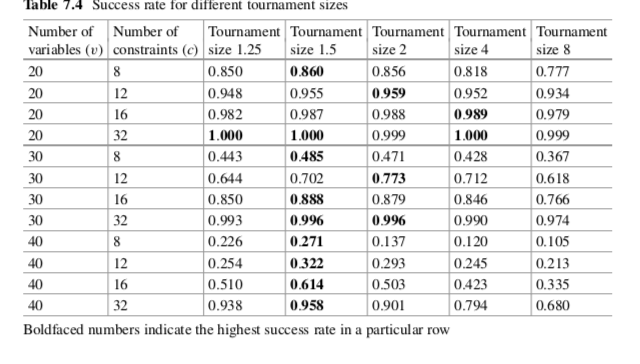
只有在所有的选择算法都接近完美成功率的情况下，改进不显著。

7.5.2按比赛规模划分的成功率

因为比赛选择的成功率更好或等于适合度比例选择，我们在本文其余部分的许多分析将只比较字典酶选择与比赛选择。此外，因为比赛选择本身是由比赛大小参数化的，所以我们进行了额外的实验来比较不同比赛大小设置下的性能。

表7.4显示了这些运行的结果。我们再次对变量数量(v)和约束数量(c)的每个组合进行3750次运行，跨越表7.1中给出的每个约束的子句数量(n)的值范围。我们对比赛大小进行了从1.25到8的测试，并按照7.4节中所述处理了非整数值的比赛大小。比赛大小为2的实验与表7.3中记录的实验是独立的，所以两个表中的数字不同，但差别不大。

从表7.4中可以看出，最有效的比赛规模是1.5或2，较大的比赛规模表现不佳。没有一种测试锦标赛大小比lexicase选择性能更好。



图形用户界面

中度可信度描述已自动生成

7.5.3进化过程中的错误

除上述成功率外，上述7.5.1节中所述的运行所产生的数据的其他特征也可以揭示。

在图7.1中，我们展示了所有运行的标准化平均误差。在这里，每个错误都通过特定运行使用的约束的总数(最大错误)来标准化。对于一个特定的代，错误已经在该代之前活跃的运行中平均了。

这里我们看到，词汇酶选择不仅产生较低的错误，而且在进化时间较早达到较低的错误。我们还看到，这两种方法都在进化早期就取得了大部分成果，在100代之后几乎没有什么改进。

7.5.4平均最小误差

图7.2显示了变量数量(v)和约束数量(c)的每个组合的最小平均误差(MLE)。最小平均误差定义为每个运行中最小误差个体的误差值的平均值:

手机屏幕截图

中度可信度描述已自动生成

图示, 工程绘图

描述已自动生成

这些图表表明，比赛选择不仅在许多情况下无法解决问题，而且在失败的情况下所取得的最佳错误往往相当高。对于每个约束有大量子句的运行，这种效果尤其明显。

7.5.5成功代

图7.3显示了成功生成的情况——即遗传算法能够找到零错误解的生成——对于变量数量(v)和约束数量(c)的每一组合，平均遗传算法能够找到解的运行情况。

在这里我们可以看到，即使使用竞赛选择的遗传算法能够找到一个解决方案，它通常也需要比使用词汇酶选择的遗传算法更多的世代来完成。

图示

描述已自动生成

7.5.6进化过程中的多样性

图7.4显示了在进化过程中种群中唯一染色体(个体)的平均数量，在所有参数组合中聚合，并按选择方法分组，以及每种方法是否找到了问题的解决方案。我们从这些情节中看到，除了在运行开始的短暂时期，lexicase选择在整个进化过程中保持相对高的多样性，无论是在成功的和不成功的运行。

7.6讨论

上述结果表明，对于本文研究的布尔约束满足问题，传统遗传算法在词汇酶亲本选择中的表现优于竞赛亲本选择或适合度比例亲本选择。在这些实验中，lexicase选择比其他亲本选择方法更频繁和更少的世代找到解决方案。

此外，在lexicase选择下，更多的种群保持了多样性，尽管这里的结果没有说任何关于可能在多样性、成功率和找到解决方案所需的世代数之间存在的因果关系。

关于这项研究的中心问题，即lexicase选择是否在遗传规划之外也有用途，这些结果给出了一个积极的答案:lexicase选择似乎比之前所证明的具有更广泛的适用性。

本研究中所研究的问题是人为的，但它们被设计成具有类似于许多现实问题的特征。更具体地说，这里研究的问题被设计成类似于现实世界的问题，其中的目标是同时满足许多约束，但约束本身及其相互依赖性都是不透明的。

对于这里研究的问题，问题解决者可以访问一个过程，该过程表明每个约束是否被候选解决方案满足，但它没有关于约束的性质或多个约束之间的共享组件或结构的其他信息。在某种程度上，现实世界的问题符合这一特性，这里的结果表明lexicase亲本选择可以帮助解决它。

尽管如此，lexicase亲本选择可能并不适用于所有的问题。例如，它似乎不太可能很好地处理只涉及单个约束的问题。在这些情况下，只有种群中在该约束下表现最好的单个个体(或其他表现相同的个体)才会被选择作为亲本。似乎有理由认为，这将破坏人口多样性，使其更难找到解决方案。由于同样的原因，有多个约束(但不多)的问题可能与lexicase父选择不匹配。

对于这里研究的问题，问题解决者可以访问一个过程，该过程表明每个约束是否被候选解决方案满足，但它没有关于约束的性质或多个约束之间的共享组件或结构的其他信息。在某种程度上，现实世界的问题符合这一特性，这里的结果表明lexicase亲本选择可以帮助解决它。

尽管如此，lexicase亲本选择可能并不适用于所有的问题。例如，它似乎不太可能很好地处理只涉及单个约束的问题。在这些情况下，只有种群中在该约束下表现最好的单个个体(或其他表现相同的个体)才会被选择作为亲本。似乎有理由认为，这将破坏人口多样性，使其更难找到解决方案。由于同样的原因，有多个约束(但不多)的问题可能与lexicase父选择不匹配。

以前的工作还表明，lexicase父选择有时在浮点错误值的问题上表现很差。epsilon lexicase选择方法似乎很好地解决了这个问题[9]，而且似乎可以合理地假设，它也可以很好地处理这里提出的约束满足问题的浮点版本。

图形用户界面, 应用程序

描述已自动生成

未来工作的一个令人兴奋的途径将是调查词汇酶选择是否有更广泛的适用性，也许会扩展到进化计算之外。其他机器学习方法也可以利用lexicase选择的核心理念,每当我们必须做出决定基于候选人的质量解决方案,而不是将多个质量指标合并为单一的标量值,我们可能考虑一次,以随机的顺序。

词汇酶选择这一核心思想的充实方式将因机器学习方法的不同而有所不同，我们还不能对进化计算以外的任何特定方法提供明确的指导。然而，本文给出的结果让我们推测，一些这样的努力将会因其所应用的机器学习方法的问题解决能力的改善而得到回报。