Statistics and Computing (1994) 4, 8 7 - 1 1 2
统计与计算(1994)4,87-112

Genetic programming as a means for programming computers by natural selection
遗传编程作为通过自然选择编程计算机的手段

JOHN R. KOZA
JOHN r. KOZA

Computer Science Department, Stanford University, Stanford, CA 94305, USA
斯坦福大学计算机科学系，斯坦福，加州 94305，美国

Many seemingly different problems in machine learning, artificial intelligence, and symbolic processing can be viewed as requiring the discovery of a computer program that produces some desired output for particular inputs. When viewed in this way, the process of solving these problems becomes equivalent to searching a space of possible computer programs for a highly fit individual computer program. The recently developed genetic programming paradigm described herein provides a way to search the space of possible computer programs for a highly fit individual computer program to solve (or approximately solve) a surprising variety of different problems from different fields. In genetic programming, populations of computer programs are genetically bred using the Darwinian principle of survival of the fittest and using a genetic crossover (sexual recombination) operator appropriate for genetically mating com-puter programs. Genetic programming is illustrated via an example of machine learning of the Boolean 11-multiplexer function and symbolic regression of the econometric exchange equation from noisy empirical data.
机器学习、人工智能和符号处理中许多看似不同的问题可以被看作是需要发现一个计算机程序，为特定的输入产生一些期望的输出。从这个角度来看，解决这些问题的过程就相当于在一个可能的计算机程序空间中搜索一个高度适合的单个计算机程序。最近开发的遗传程序设计范式描述在这里提供了一种方法，搜索可能的计算机程序的空间为一个高度适合的个人计算机程序解决(或近似解决)来自不同领域的各种令人惊讶的不同问题。在遗传程序设计中，计算机程序的种群是按照达尔文的适者生存原则进行遗传繁殖的，并使用适合于计算机程序遗传交配的遗传交叉(性重组)算子。遗传编程是通过一个例子来说明的布尔 11-多工函数的机器学习和符号回归的经济计量交换方程从嘈杂的经验数据。

Hierarchical automatic function definition enables genetic programming to define potentially useful functions automatically and dynamically during a run, much as a human programmer writing a complex computer program creates subroutines (procedures, functions) to perform groups of steps which must be performed with different instantiations of the dummy variables (formal parameters) in more than one place in the main program. Hierarchical automatic func-tion definition is illustrated via the machine learning of the Boolean 11-parity function.

层次自动函数定义使遗传编程能够在运行过程中自动和动态地定义潜在有用的函数，就像人类编程人员编写一个复杂的计算机程序创建子程序(过程，函数)来执行一组步骤，这些步骤必须在主程序的多个地方通过虚变量(形式参数)的不同实例化来执行。层次化自动函数定义是通过布尔 11 奇偶函数的机器学习来说明的。

Keywords. Genetic programming, genetic algorithm, crossover, hierarchical automatic func-tion definition, symbolic regression, Boolean 11-multiplexer, econometric exchange equation, Boolean 11-parity
关键词: 遗传规划、遗传算法、交叉、层次自动函数定义、符号回归、布尔 11 复用器、计量经济交换方程、布尔 11 奇偶校验

I. Introduction and overview
引言和概述

C o m p u t e r program s are a m o n g the most complex and intricate structures created by h u m a n beings. They are usually written line by line by applying h u m a n knowledge and intelligence to the proble m at hand. Writing a com - puter p r o g r a m is usually difficult. Indeed, one o f the central questions in compute r science (attributed to Arthu r Samuel in the 1950s) is
程序是人类创造的最复杂、最复杂的结构。它们通常是通过将知识和智慧应用到手头的问题上，一行一行地写出来的。写一个计算机文档通常是很困难的。事实上，计算机科学的核心问题之一(20 世纪 50 年代由 arthur r Samuel 提出)就是

H o w can computers learn to solve problems without being explicitly p r o g r a m m e d ? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?
计算机如何才能学会解决问题而不需要明确的求解呢？换句话说，如何才能让计算机做需要做的事情，而不被告知到底该怎么做？

In the natural world, complex and intricate structures do not arise via explicit design and p r o g r a m m i n g or fro m the application o f h u m a n intelligence. Instead, complex and successful organic structures evolve over a period of time as the consequence o f Darwinian natural selection and the creative effects o f sexual recombination (genetic cross-over) and mutation . Complex structures evolve in nature as a consequence o f a fitness metric applied by the problem environment because structures that are mor e fit in grap-pling with their environment survive and reproduce at a higher rate.

在自然世界中，复杂而复杂的结构不是通过明确的设计和设计或 0960-317491994 Chapman & Hall 产生的人类智慧的应用。相反，由于达尔文的自然选择和性重组(遗传交叉)和突变的创造性影响，复杂而成功的有机结构在一段时间内不断进化。在自然界中，复杂结构的演化是问题环境应用适应性度量的结果，因为更适合与其环境相适应的结构以更高的速度生存和繁殖。

The question arises as to whether an analogue o f natural selection and genetics can be applied to the proble m o f creating a p r o g r a m that enables a computer to solve a problem . Tha t is, can complex compute r program s be created, not via h u m a n intelligence, but by applying a fit-ness measure appropriate to the proble m environment?
问题是，自然选择和遗传学的类比是否可以应用于创造一个计算机能够解决问题的方法。也就是说，复杂的计算机程序是否可以创建，不是通过人工智能，而是通过适用于问题环境的适应性度量？

Such a process o f genetical breeding o f compute r pro -
这是一个遗传育种的过程

grams might start with a primordial ooze consisting of a population of hundreds or thousands of randomly created computer programs of various randomly determined sizes and shapes. In such a process, each program in the popu-lation would be observed as it tries to grapple with its environment--that is, to solve the problem at hand. A value would then be assigned to each program, reflecting how fit it is in solving the problem at hand. We might then allow a program in the population to survive to a later generation of the process with a probability propor-tionate to its observed fitness. Additionally, we might also select pairs of programs from the population with a prob-ability proportionate to their observed fitness and create new offspring by recombining subprograms from them at random. We would apply the above steps to the popu-lation of programs over a number of generations.
克可能从一个由数百或数千个随机创建的计算机程序组成的原始软泥开始，程序的大小和形状都是随机确定的。在这样一个过程中，人口中的每个程序都会被观察到，因为它试图与其环境作斗争——也就是说，解决手头的问题。然后给每个程序分配一个值，反映它在解决手头问题上的适合度。然后，我们可能允许一个程序在人口中存活到下一代的过程中，其观察到的适应性与概率成比例。此外，我们也可以从人口中选择与其观察到的适应性成比例的概率程序对，并通过随机重组子程序来创建新的后代。我们将把上述步骤应用于几代人的程序人口。

Anyone who has ever written and debugged computer programs and has experienced their brittle, highly non-linear, and perversely unforgiving nature will probably be understandably sceptical about the proposition that the biologically motivated process sketched above could possibly produce a useful computer program. However, in this article we present a number of examples from various fields supporting the surprising and counter-intuitive notion that computers can indeed be programmed by means of natural selection. We will show, via examples, that the recently developed genetic programming paradigm provides a way to search the space of all possible programs to find a function which solves, or approximately solves, a problem.
任何曾经编写和调试过计算机程序，并且经历过它们脆弱、高度非线性和顽固不饶人的天性的人，都可以理解地怀疑上面所描述的生物动机过程可能会产生一个有用的计算机程序这一命题。然而，在这篇文章中，我们提出了许多来自不同领域的例子来支持这个令人惊讶的和反直觉的概念，即计算机确实可以通过自然选择来编程。我们将通过例子说明，最近开发的遗传编程范型提供了一种搜索所有可能的程序空间的方法，以找到解决或近似解决问题的函数。

2. Background on genetic algorithms and genetic programming
遗传算法和遗传编程的背景

John Holland's pioneering book *Adaptation in Natural and Artificial Systems* described how the evolutionary process in nature can be applied to artificial systems using the genetic algorithm operating on fixed-length character strings (Holland, 1975). Holland demonstrated that a population of fixed-length character strings (each repre-senting a proposed solution to a problem) can be gene-tically bred using the Darwinian operation of fitness proportionate reproduction and the genetic operation of recombination. The recombination operation combines parts of two chromosome-like fixed-length character strings, each selected on the basis of their fitness, to pro-duce new offspring strings. Holland established, among other things, that the genetic algorithm is a near optimal approach to adaptation in that it maximizes expected over-all average payoff when the adaptive process is viewed as a multi-armed slot machine problem requiring an optimal allocation of future trials given currently available informa-tion. The genetic algorithm has proven successful at search-ing non-linear multidimensional spaces in order to solve, or
约翰 ·霍兰德的开创性著作《自然和人工系统中的适应》描述了自然界中的进化过程是如何应用到人工系统中的，使用的是基于固定长度字符串的遗传算法(Holland，1975)。Holland 证明了一个固定长度的字符串群体(每个字符串代表一个问题的建议解决方案)可以通过达尔文的适应性比例繁殖运算和重组的遗传运算进行基因繁殖。重组操作将两个类似染色体的固定长度字符串的部分组合在一起，每个字符串都是根据其适应性选择的，以产生新的后代字符串。Holland 认为，遗传算法是一种近乎最优的适应方法，因为当自适应过程被视为一个多臂老虎机问题，需要根据目前可获得的信息对未来试验进行最优分配时，遗传算法可以使预期的总体平均收益最大化。遗传算法已被证明在搜索非线性多维空间以解决，或

approximately solve, a wide variety of problems (Goldberg, 1989; Davis, 1987; 1991; Davidor, 1991; Michalewicz, 1992). Recent conference proceedings provide an overview of current work in the field (Schaffer, 1989; Forrest, 1990; Belew and Booker, 1991; Rawlins, 1991; Mayer and Wilson, 1991; Schwefel and Maenner, 1991; Langton *et al.*, 1992; Whitley, 1992).
近似地解决各种各样的问题(Goldberg，1989; Davis，1987; 1991; Davidor，1991; Michalewicz，1992)。最近的会议记录提供了该领域当前工作的概述(Schaffer，1989; Forrest，1990; Belew 和 Booker，1991; Rawlins，1991; Mayer 和 Wilson，1991; Schwefel 和 Maenner，1991; Langton 等，1992; Whitley，1992)。

Representation is a key issue in genetic algorithm work because genetic algorithms directly manipulate a coded chromosomal representation of the problem. The representation scheme can therefore severely limit the window by which the system observes its world. On the other hand, the use of fixed-length character strings has permitted Holland and others to construct a significant body of theory as to why genetic algorithms work. Much of this theoretical analysis depends on the mathematical tract-ability of the fixed-length character strings as compared with mathematical structures that are more complex and comparatively less susceptible to theoretical analysis. The need for increasing the complexity of the structures under-going adaptation using the genetic algorithm has been reflected by considerable work over the years in that direc-tion (Smith, 1980; Cramer, 1985; Holland, 1986; Holland *et al.*, 1986; Wilson, 1987a,b; Fujiki and Dickinson, 1987; Goldberg *et al.*, 1989).

表示是遗传算法工作中的一个关键问题，因为遗传算法直接操纵问题的编码染色体表示。因此，表示方案可以严重限制系统观察其世界的窗口。另一方面，固定长度字符串的使用使 Holland 和其他人能够构建一个关于遗传算法工作原理的重要理论体系。这种理论分析在很大程度上依赖于定长字符串的数学牵引能力，而相比之下，定长字符串的数学结构更为复杂，对理论分析的敏感性相对较低。多年来在这方面进行的大量工作反映了利用遗传算法增加正在适应的结构的复杂性的必要性(Smith，1980; Cramer，1985; Holland，1986; Holland 等，1986; Wilson，1987a，b; Fujiki 和 Dickinson，1987; Goldberg 等，1989)。

For many problems in machine learning and artificial intelligence, the most natural representation for a solution is a computer program (i.e. a hierarchical composition of primitive functions and terminals) of indeterminate size and shape, as opposed to character strings whose size has been determined in advance. It is difficult, unnatural, and overly restrictive to attempt to represent hierarchies of dynamically varying size and shape with fixed-length char-acter strings.

对于机器学习和人工智能中的许多问题，解决方案最自然的表示形式是大小和形状不确定的计算机程序(即由原始函数和终端组成的分层结构)，而不是事先确定大小的字符串。试图用固定长度的字符串来表示动态变化的大小和形状的层次结构是困难的，不自然的，而且过于严格的。

Genetic programming provides a way to find a computer program of unspecified size and shape to solve, or approxi-mately solve, a problem. The book *Genetic Programming." On the Programming of Computers by Means of Natural Selection* (Koza, 1992a) describes genetic programming in detail. A videotape visualization of applications of genetic programming can be found in the *Genetic Programming: The Movie* (Koza and Rice, 1992a). See also Koza (1992b).

遗传编程提供了一种方法，找到一个大小和形状不明确的计算机程序来解决，或近似解决，一个问题。《遗传编程》一书关于通过自然选择来编程计算机(Koza，1992a)详细描述了遗传编程。遗传编程应用的录像带可以在《遗传编程: 电影》(Koza and Rice，1992a)中找到。参见 Koza (1992b)。

3. Overview of genetic programming
3. 基因编程概述

Genetic programming continues the trend of dealing with the problem of representation in genetic algorithms by increasing the complexity of the structures undergoing adaptation. In particular, the individuals in the population in genetic programming are hierarchical compositions of primitive functions and terminals appropriate to the parti-cular problem domain. The set of primitive functions used typically includes arithmetic operations, mathematical functions, conditional logical operations, and domain-
遗传规划通过增加正在进行适应的结构的复杂性，继续处理遗传算法中的表示问题的趋势。特别是，遗传编程中人群中的个体是适合于特定问题领域的原始函数和终端的分层组成。通常使用的基本函数集包括算术运算、数学函数、条件逻辑运算和域-

specific functions. The set o f terminals used typically includes inputs appropriate to the problem domain and various numeric constants.
使用的一组终端通常包括与问题域相适应的输入和各种数值常量。

The compositions of primitive functions and terminals described above correspond directly to the computer pro-grams found in programming languages such as LISP (where they are called symbolic expressions or S-expres-sions). An S-expression can be represented as a rooted, point-labelled tree with ordered branches in which the root and other internal points of the tree are labelled with functions and in which the external points of the tree are labelled with terminals. In fact, these compositions corres-pond directly to the parse tree that is internally created by the compilers of most programming languages. Thus, genetic programming views the search for a solution to a problem as a search in the space of all possible com-positions of functions that can be recursively composed o f the available primitive functions and terminals.

上面描述的基本函数和终端的组合直接对应于 LISP 等编程语言中的计算机程序(在那里它们被称为符号表达式或 s 表达式)。一个 s 表达式可以表示为一个带有有序分支的有根的点标记树，树的根和其他内部点用函数标记，树的外部点用终端标记。事实上，这些组合直接对应于大多数编程语言的编译器在内部创建的解析树。因此，遗传程序设计将寻找问题的解决方案视为在可以由可用的基本函数和终端递归组成的函数的所有可能组合空间中的搜索。

O f course, virtually any problem in artificial intelligence, symbolic processing, and machine learning can be viewed as requiring discovery of a computer program that produces some desired output for particular inputs. The process of solving these problems can be reformulated as a search for a highly fit individual computer program in the space of possible computer programs. When viewed in this way, the process of solving these problems becomes equivalent to searching a space o f possible computer programs for the fittest individual computer program. In particular, the search space is the space of all possible computer programs composed of functions and terminals appropriate to the problem domain. Genetic programming provides a way to search for this fittest individual computer program.

当然，实际上人工智能、符号处理和机器学习中的任何问题都可以被看作是需要发现一个计算机程序，为特定的输入产生一些期望的输出。解决这些问题的过程可以被重新表述为在可能的计算机程序空间中寻找一个高度适合的个人计算机程序。以这种方式来看，解决这些问题的过程就相当于在一个可能的计算机程序空间中搜索最适合的个人计算机程序。特别地，搜索空间是所有可能的计算机程序的空间，它由适合问题域的函数和终端组成。遗传编程提供了一种搜索最适合的个人计算机程序的方法。

In genetic programming, populations of hundreds or thousands o f computer programs are genetically bred. This breeding is done using the Darwinian principle of sur-vival and reproduction of the fittest along with a genetic recombination (crossover) operation appropriate for mating computer programs. As will be seen, a computer program that solves (or approximately solves) a given prob - lem may emerge from this combination of Darwinian natural selection and genetic operations.

在基因编程中，成百上千的计算机程序是通过基因繁殖的。这种繁殖是使用达尔文的生存和适者生殖原则，以及适合于交配的计算机程序的遗传重组(交叉)操作。正如我们将看到的，一个能够解决(或者近似解决)一个给定问题的计算机程序可能会从达尔文自然选择和遗传操作的结合中产生。

Genetic programming starts with an initial population o f randomly generated computer programs composed of func-tions and terminals appropriate to the problem domain. The functions may be standard arithmetic operations, standard programming operations, standard mathemati-cal functions, logical functions, or domain-specific func-tions. Depending on the particular problem, the computer program may be Boolean-valued, integer-valued, real-valued, complex-valued, vector-valued, symbolic-valued, or multiple-valued. The creation of this initial rando m population is, in effect, a blind rando m search of the search space of the problem.

遗传编程从随机生成的计算机程序的初始种群开始，这些程序由适合于问题域的函数和终端组成。函数可以是标准算术运算，标准编程运算，标准数学函数，逻辑函数或领域特定的函数。根据具体问题的不同，计算机程序可以是布尔值、整数值、实数值、复数值、向量值、符号值或多值。这个初始随机 m 种群的产生实际上是对问题搜索空间的盲目随机 m 搜索。

Each individual computer program in the population is measured in terms o f how well it performs in the particular
群体中的每个单独的计算机程序都是根据它在特定情况下的表现来衡量的

problem environment. This measure is called the *fitness*
这种方法被称为健身
measure.
措施。

The nature o f the fitness measure varies with the prob - lem. Fo r man y problems, fitness is naturally measured by the error produced by the computer program. The closer this error is to zero, the better the computer program. If one is trying to find a good randomizer, the fitness of a given computer program might be measured via entropy. The higher the entropy, the better the randomizer. If one is trying to recognize patterns or classify examples, the fit-ness of a particular program might be the number of examples (instances) it handles correctly. The more examples correctly handled, the better. In a problem of optimal control, the fitness of a computer program may be the amoun t of time or fuel or money required to bring the system to a desired target state. The smaller the amount of time or fuel or money, the better. F o r some problems, fitness ma y consist of a combination of factors such as correctness, parsimony, or efficiency.

健身措施的性质随问题的不同而不同。对于人类的问题，健康自然是通过计算机程序产生的误差来衡量的。这个误差越接近零，计算机程序就越好。如果一个人试图找到一个好的随机发生器，一个给定的计算机程序的适应性可以通过熵来衡量。熵越高，随机选择器就越好。如果一个人试图识别模式或对示例进行分类，那么一个特定程序的适用性可能是它正确处理的示例(实例)的数量。正确处理的例子越多越好。在最优控制问题中，计算机程序的适应性可能是使系统达到预期目标状态所需的时间、燃料或金钱。时间、燃料或金钱越少越好。对于某些问题，健康可能由诸如正确性、节俭或效率等因素组成。

Typically, each computer program in the population is run over a number of different *fitness cases* so that its fit-ness is measured as a sum or an average over a variety of representative different situations. These fitness cases some-times represent a sampling of different values o f an indepen-dent variable or a sampling of different initial conditions of

通常情况下，人口中的每个计算机程序都运行在许多不同的适应性情况下，因此其适应性是以各种具有代表性的不同情况下的总和或平均值来衡量的。这些适应性案例有时代表了一个独立变量的不同值的抽样或一个不同初始条件的抽样

a system. F o r example, the fitness of an individual com-puter program in the population may be measured in terms of the sum of the squares of the differences between the output produced by the program and the correct answer to the problem. This sum may be taken over a samp-ling of different inputs to the program. The fitness cases may be chosen at rando m or may be structured in some way.

系统。例如，单个计算机程序在总体中的适应性可以用程序产生的输出与问题正确答案之间差异的平方和来衡量。这个总和可以通过对程序不同输入的抽样得到。适应性案例可以在随机 m 中选择，也可以以某种方式构造。

The Computer programs in generation 0 will have exceed-ingly p o o r fitness. Non e the less, some individuals in the population will turn out to be somewhat fitter than others. These differences in performance are then exploited. The Darwinian principle o f reproduction and survival of the fittest and the genetic operation of sexual recombination (crossover) are used to create a new off-spring population of individual computer programs from the current population o f programs. The reproduction operation involves selecting on the basis of fitness (i.e. the fitter the program, the more likely it is to be selected), a computer program from the current population of pro-grams, and allowing it to survive by copying it into the new population.

第 0 代计算机程序的适应性将超过极限。尽管如此，人口中的某些个体还是会比其他个体更健康。这些表现上的差异会被利用。利用达尔文的适者生存和繁殖原理以及有性重组的遗传操作(交叉)，从现有的计算机程序种群中创建了一个新的个体计算机程序的后代种群。复制操作包括基于适应性(即程序越适合，被选择的可能性就越大)从当前的程序种群中选择一个计算机程序，并通过将其复制到新的种群中来使其存活。

The genetic process o f sexual reproduction between two parental computer programs is used to create new off-spring computer programs from two parental programs selected on the basis of fitness. The parental programs are typically of different sizes and shapes. The offspring pro-grams are composed of subexpressions (subtrees, subpro-grams, subroutines, building blocks) from their parents. These offspring programs are typically o f different sizes

利用两个亲本计算机程序之间有性生殖的遗传过程，从基于适应性选择的两个亲本程序中创建新的子代计算机程序。亲本程序通常具有不同的大小和形状。子代程序由来自父母的子表达式(子树、子程序、子程序、构建块)组成。这些后代程序通常具有不同的大小

and shapes than their parents. Intuitively, if two computer programs are somewhat effective in solving a problem, then some of their parts probably have some merit. By recom-bining randomly chosen parts of somewhat effective programs, we may produce new computer programs that are even fitter in solving the problem. For example, con-sider the following computer program (LISP symbolic expression):
比他们的父母和形状。直观地说，如果两个计算机程序在解决一个问题上有一定的效率，那么它们的一些部分可能有一些优点。通过重新组合随机选择的部分有效的程序，我们可以生产出更适合解决问题的新的计算机程序。例如，考虑下面的计算机程序(LISP 符号表达式) :

(+('0 . 234Z) (-X0.789)),
(+ (’0.234Z)(- X0.789))

which we would ordinarily write as
我们通常会写成

0.234Z + X - 0.789.
0.234 z + x-0.789.

This program takes two inputs (X and Z) and produces a floating-point output. In the prefix notation used, the multi-plication function * is first applied to the terminals 0.234 and Z to produce an intermediate result. Then, the subtrac-tion function - is applied to the terminals X and 0.789 to produce a second intermediate result. Finally, the addition function + is applied to the two intermediate results to pro-duce the overall result.
这个程序接受两个输入(x 和 z)并产生一个浮点输出。在使用的前缀表示法中，乘法函数 * 首先应用于 0.234 和 z 终端，以产生一个中间结果。然后，减法函数-应用于终端 x 和 0.789 产生第二个中间结果。最后，将加法函数 + 应用于两个中间结果，以产生整体结果。

Also, consider a second program:
另外，考虑第二个程序:

(* (*ZY) (+Y(*0.314Z))),
(* (* ZY)(+ y (* 0.314 z)) ,

which is equivalent to
相当于

ZY(Y + 0.314Z).
ZY (y + 0.314 z).

In Fig. 1, these two programs are depicted as rooted, point-labelled trees with ordered branches. Internal points (i.e. nodes) of the tree correspond to functions (i.e. opera-tions) and external points (i.e. leaves, endpoints) corre-spond to terminals (i.e. input data). The numbers beside the function and terminal points of the tree appear for refer-ence only. The crossover operation creates new offspring by exchanging subtrees (i.e. sublists, subroutines, subproce-dures) between the two parents.
在图 1 中，这两个程序被描绘为带有有序分支的根，点标记的树。树的内部点(即节点)对应于函数(即操作) ，而外部点(即叶子、端点)对应于终端(即输入数据)。树的函数和终端点旁边的数字仅供参考。交叉操作通过在两个父树之间交换子树(即子列表，子程序，子程序)来创建新的子树。

Assume that the points of both trees are numbered in a depth-first way starting at the left. Suppose that the point number 2 (out of 7 points of the first parent) is randomly selected as the crossover point for the first parent and that the point number 5 (out of 9 points of the second parent) is randomly selected as the crossover point of the
假设两棵树的点都是从左边开始以深度优先的方式编号的。假设随机选择第 2 个点(第一亲本的 7 个点中的 7 个点)作为第一亲本的交叉点，随机选择第 5 个点(第二亲本的 9 个点中的 9 个点中的 5 个)作为第一亲本的交叉点

0.234Z Y + 0.314Z
0.234 z y + 0.314 z

Fig. 2. *Two crossover fragments*
图 2. 两个交叉片段

second parent. The crossover points in the trees above are therefore the * in the first parent and + in the second parent. The two crossover fragments are the two sub-trees shown in Fig. 2. These two crossover fragments corres-pond to the underlined subprograms (sublists) in the two parental computer programs. The two offspring resulting from crossover are
第二个家长。因此，上面树中的交叉点是第一个父类中的 * 和第二个父类中的 + 。两个交叉片段是图 2 所示的两个子树。这两个交叉片段对应于两个父母计算机程序中带下划线的子程序(子列表)。由交叉产生的两个后代是

(+(+Y(*0.314Z)) (-X0.789))
(+ (+ y (* 0.314 z))(- X0.789))

and
还有

(* (*ZY)(*0.234Z)).
(* (* ZY)(* 0.234 z)).

The two offspring are shown in Fig. 3.
这两个后代如图 3 所示。

Thus, crossover creates new computer programs using parts of existing parental programs. Because entire sub-trees are swapped, this crossover operation always pro-duces syntactically and semantically valid programs as offspring regardless of the choice of the two crossover points. Because programs are selected to participate in the crossover operation with a probability proportional to fitness, crossover allocates future trials to areas of the search space represented by programs containing parts from promising programs.
因此，交叉使用现有父母程序的一部分创建新的计算机程序。因为整个子树是交换的，所以这种交叉操作总是产生语法和语义上有效的程序作为后代，而不管这两个交叉点的选择如何。因为程序被选择参与交叉操作的概率与适应度成正比，交叉将未来的试验分配给搜索空间的区域，这些区域由包含有希望程序的部分的程序所代表。

After the operations of reproduction and crossover are performed on the current population, the population of off-spring (i.e. the new generation) replaces the old population (i.e. the old generation). Each individual in the new popu-lation of computer programs is then measured for fitness, and the process is repeated over many generations.
在对当前种群进行繁殖和交配操作后，后代(即新一代)的种群取代了旧种群(即旧一代)。然后，计算机程序的新种群中的每个个体都会被测量其适应性，这个过程会在许多代中重复。

At each stage of this highly parallel, locally controlled,
在这个高度平行，局部控制的过程的每个阶段，

0.234Z + X - 0.789 ZY(Y + 0.314Z)
0.234 z + x-0.789 ZY (y + 0.314 z)

Fig. 1. *Two parental computer programs*
图 1。两个父母计算机程序。图 3。两个后代

0.234Z2
Y
Y + 0.314Z + X - 0.789 0.234
Y + 0.314 z + x-0.789 Z2Y

Fig. 3. *Two offspring*

decentralized process, the state of the process will consist only of the current population of individuals. The force driving this process consists only of the observed fitness of the individuals in the current population in grappling with the problem environment.
分散的过程，过程的状态将只包括当前的人口的个人。推动这一进程的力量只包括观察到的当前人口中的个体在应对问题环境中的适应性。

As will be seen, this algorithm will produce populations of computer programs which, over many generations, tend to exhibit increasing average fitness in dealing with their environment. In addition, these populations of computer programs can rapidly and effectively adapt to changes in the environment. Typically, the best individual that appeared in any generation of a run (i.e. the best-so-far individual) is designated as the result produced by genetic programming.

正如我们将看到的，这种算法将产生大量的计算机程序，经过许多代，这些程序在处理环境时往往表现出越来越强的平均适应性。此外，这些计算机程序群可以迅速有效地适应环境的变化。一般来说，在任何一代中出现的最好的个体(即迄今为止最好的个体)被指定为遗传编程产生的结果。

The hierarchical character of the computer programs that are produced is an important feature of genetic programming. The results of genetic programming are inherently hierarchical. In many cases the results produced by genetic programming are default hierarchies, prioritized hierarchies of tasks, or hierarchies in which one behaviour subsumes or suppresses another.

生成的计算机程序的层次性是遗传程序设计的一个重要特征。遗传编程的结果本质上是等级的。在许多情况下，遗传编程产生的结果是默认的层次结构，任务的优先级层次结构，或者一种行为包含或抑制另一种行为的层次结构。

The dynamic variability of the computer programs that are developed along the way to a solution is also an impor-tant feature of genetic programming. It would be difficult and unnatural to try to specify or restrict the size and shape of the eventual solution in advance. Moreover, advance specification or restriction of the size and shape of the solution to a problem narrows the window by which the system views the world and might well preclude finding the solution to the problem at all.

计算机程序在解决问题的过程中的动态变化也是遗传编程的一个重要特征。预先设定或限制最终解决方案的大小和形状是困难和不自然的。此外，提前说明或限制问题解决方案的大小和形状会缩小系统观察世界的窗口，很可能根本无法找到问题的解决方案。

Another important feature of genetic programming is the absence or relatively minor role of preprocessing of inputs and postprocessing of outputs. The inputs, intermediate results, and outputs are typically expressed directly in terms of the natural terminology of the problem domain. The computer programs produced by genetic program-ming consist of functions that are natural for the problem domain.

遗传规划的另一个重要特点是没有投入的预处理和产出的后处理，或者这种作用相对较小。输入，中间结果和输出通常直接用问题领域的自然术语表示。遗传编程产生的计算机程序由问题域自然的函数组成。

Finally, the structures undergoing adaptation in genetic programming are active. They are not passive chromo-somal encodings of the solution to the problem. Instead, given a computer on which to run, the structures in genetic programming are active program structures that are cap-able of being executed in their current form.

最后，在遗传编程中进行适应的结构是活跃的。它们不是解决问题的被动染色体编码。相反，给定了一台运行的计算机，遗传编程中的结构是活跃的程序结构，能够以其当前形式执行。

In summary, genetic programming breeds computer pro-grams to solve problems by executing the following three steps:

总之，遗传编程通过执行以下三个步骤培育计算机程序来解决问题:

1. Generate an initial population of random computer programs composed of the primitive functions and terminals of the problem.
生成由问题的基本函数和终端组成的随机计算机程序的初始种群。
2. Iteratively perform the following substeps until the termination criterion for the run has been satisfied:
迭代地执行以下子步骤，直到满足运行的终止标准为止:
   (a) Execute each program in the population so that a fitness measure indicating how well the program
   在总体中执行每个程序，以便一个适应性度量指示程序的好坏

solves the problem can be computed for the program.
解决问题的程序可以计算出来。

(b) Create a new population of programs by selecting program(s) in the population with a probability based on fitness (i.e. the fitter the program, the more likely it is to be selected) and then applying the following primary operations:
创建一个新的程序群体，方法是在群体中选择基于适应性概率的程序(即程序越适合，被选择的可能性就越大)，然后应用以下主要操作:

(i) *Reproduction:* Copy an existing program to the new population.
复制: 将现有程序复制到新的人口。

(ii) *Crossover:* Create two new offspring pro-grams for the new population by genetically recombining randomly chosen parts of two existing programs.
交叉: 通过基因重组两个现有程序中随机选择的部分，为新种群创建两个新的后代程序。

3. The single best computer program in the population produced during the run is designated as the result of the run of genetic programming. This result may be a solution (or approximate solution) to the problem.
在运行期间产生的群体中最好的一个计算机程序被指定为运行遗传编程的结果。这个结果可能是问题的一个解决方案(或近似解决方案)。

Figure 4 is a flowchart for genetic programming. The index i refers to an individual in the population of size M. The variable GEN is the number of the current generation. The box labelled 'evaluate fitness of each indivi-dual in the population' typically consumes the vast majority of computer resources.
图 4 是遗传编程的流程图。指数 i 是指大小为 m 的人群中的个体。变量 GEN 是当前一代的数量。标有“评估每个人的适应性-人口中的双重”的框通常消耗绝大多数计算机资源。

In the remainder of this article, we illustrate genetic programming with several examples chosen to illustrate various different categories of problems, namely:
在本文的其余部分，我们用几个选择的例子来说明遗传编程，以说明各种不同类别的问题，即:

9 symbolic regression of a Boolean-valued function;
布尔值函数的符号回归;
9 symbolic regression of noisy numeric-valued empirical data;
噪声数值经验数据的符号回归;
9 a multidimensional control problem; 9 a classification problem; 9 a robotics problem;
9 多维控制问题; 9 分类问题; 9 机器人问题;
9 a problem employing hierarchical automatic function definition.
9 采用分层自动函数定义的问题。

4. Symbolic regression-ll-multiplexer 符号回归 -ll-多路复用器

The problem of symbolic function identification (symbolic regression) requires developing a composition of terminals and functions that can return the correct value of the func-tion after seeing a finite sampling of combinations of the independent variable associated with the correct value of the dependent variable. The problem of machine learning of a Boolean function is a special case of symbolic regres-sion in which the independent variables are Boolean-valued, the functions being composed are Boolean func-tions, and the dependent variable is Boolean-valued.
符号函数识别(符号回归)问题需要开发终端和函数的组合，在看到与因变量的正确值相关联的自变量组合的有限取样后，能够返回函数的正确值。布尔函数的机器学习问题是一种特殊的符号回归问题，其中自变量为布尔值，所组成的函数为布尔函数，因变量为布尔值。

The problem of learning the Boolean l 1-multiplexer function will serve to show the interplay in genetic pro-gramming of
学习布尔 l1-多路复用器函数的问题将有助于显示遗传编程中的相互作用

9 the genetic variation inevitably created in the initial random generation;
9 初始随机世代不可避免地产生遗传变异;

Fig. 4. *Flowchart for genetic programming*
图 4 遗传程序设计流程图

[G e n : = 0 [
[格林: = 0]

Create Initial [
创建初始化
[Random ;opulation,
[随机; 人口,

Y e s .
是的。
. ~ f Termination ~2.~lDeslgnatel
. ~ f Termination ~ 2. ~ lDeslgnatel
~C r k e r i o n S a t i s f i e d y - I Resttlt [
C r k e r i o n s a t i s f i e d y-i Resttlt [
No4 ,
四号,

Evaluate fitness of each **I**
评估每个 i 的适应性
individual in population !
人口中的个体！

[G e n : = G e n + I ~ ~
[g e n: = g e n + i ~ ~

i' No
我没有

Select Genetic Operation'~
选择基因操作' ~
Probabilistically ,ff
概率上来说

Select One [ ] Select Two Individuals **I**
选择一个[]选择两个人
Individual] [Based on Fitness t
个人][基于 Fitness t
Based on Fitness[
基于健康[

[Perform Reproduction] [i := 1 [
[进行复制][i: = 1]

. k
. k

Perform
表演
Crossover [
交叉

Insert **T w o**
插入武器
.C spr'mg
C spr' mg
int o New
全新的
Population
人口

9 the small improvements for some individuals in the population via localized hill-climbing from generation to generation;

9 通过一代又一代的局部爬山，人群中的一些个体有了小的改善;

9 the way particular individuals become specialized and able to handle correctly certain subcases o f the prob - lem (case-splitting);

特定个体变得专业化并能够正确处理问题的某些子案例的方式(案例分裂) ;

9 the creating role o f crossover in recombining valuable parts o f more fit parents;

9 交叉在重新组合更合适的父母的有价值的部分中的创造作用;

9 how the nurturing o f a large population o f alternative solutions to the problem (rather than a single point in the solution space) helps avoid false peaks in the search for the solution to the problem;

9 培养大量可供选择的问题解决方案(而不是解决方案空间中的单一点)如何有助于避免在寻找问题解决方案时出现错误的峰值;

9 that it is not necessary to determine in advance the size and shape o f the ultimate solution or the intermediate results that ma y contribute to the solution.

没有必要事先确定最终解的大小和形状，也没有必要事先确定可能有助于解的中间结果。

The input to the Boolean N-multiplexer function consists o f k address bits ai and 2 k data bits di, where N = k + 2 k.

布尔 n 复用器函数的输入包括 f k 地址位 ai 和 2 k 数据位 di，其中 n = k + 2 k。

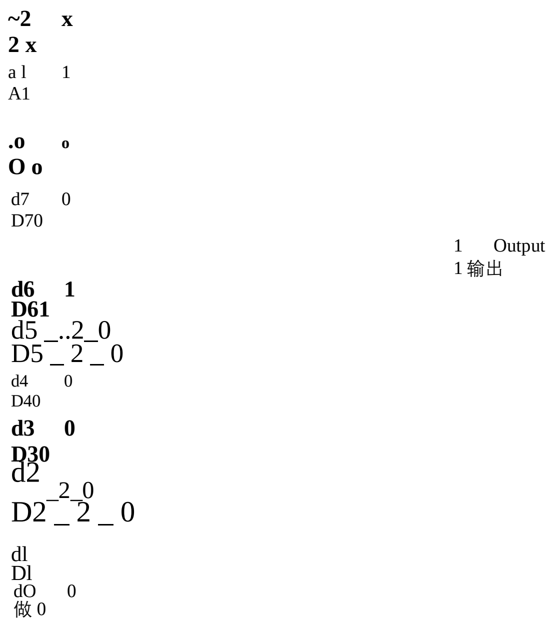

Fig. 5. *Boolean l l-multiplexer*
图 5。布尔多路复用器

That is, the input consists o f the k + 2 k bits
也就是说，输入由 k + 2 k 位组成

a k - 1 , . . . , *al*, a0, d2k_l, 99 9 dl, do.
A k-1，... ，al，a0，d2k _ l，999 dl，do.

The value o f the Boolean multiplexer function is the Boolean value (0 or 1) o f the particular data bit that is singled out by the k address bits o f the multiplexer. Fo r example, for the Boolean 11-multiplexer (where k = 3), if the three address bits *a2alao* are 110, the multiplexer singles out data bit number 6 (i.e. d6) to be the output o f the multiplexer. Figure 5 shows a Boolean 11-multiplexer with an input o f 11001000000 and the corresponding output o f 1.
布尔复用器函数的值为特定数据位的布尔值(0 或 1) o，该数据位由复用器的 k 地址位选出。例如，对于布尔 11 多路复用器(其中 k = 3) ，如果三个地址位 a2alao 为 110，多路复用器将数据位数 6(即 d6)挑选出来作为多路复用器的输出。图 5 显示了一个输入为 11001000000，输出为 1 的布尔 11 复用器。

There are five major steps in preparing to use genetic programming, namely determining:
准备使用遗传程序设计有五个主要步骤，即确定:

1. the set o f terminals;
终端设置;
2. the set o f primitive functions;
原始函数集;
3. the fitness measure;
健身措施;
4. the parameters for controlling the run;
控制跑步的参数;
5. the method for designating a result and the criterion for terminating a run.
指定结果的方法和终止运行的标准。

The first major step in preparing to use genetic programming is the identification o f the set o f terminals that will be available for constructing the computer programs (S-expressions) that will try to solve the problem. This choice is especially straightforward for this problem. The terminal set for this problem consists o f the 11 inputs to the Boolean 11-multiplexer. Thus, the terminal set Y- for this problem consists o f
准备使用遗传程序的第一个主要步骤是识别一组终端，这些终端可用于构建试图解决这个问题的计算机程序(s 表达式)。这个选择对于这个问题来说特别简单。这个问题的终端设置由布尔 11-multiplexer 的 11 个输入组成。因此，这个问题的终端集 y-由 f 组成

~'- = {A0, AI, A2, DO, D 1 , . . . , D7}.
~ ’-= { A0，AI，A2，DO，d1，. . ，D7}。

The second major step in preparing to use genetic programming is the identification of a sufficient set o f primitive functions that will be available for constructing the computer programs (S-expressions) that solve the prob-lem. Thus, the function set ~ for this problem is
准备使用遗传程序设计的第二个主要步骤是确定一组足够的基本函数，这些基本函数可用于构造解决这个问题的计算机程序(s 表达式)。因此，这个问题的函数集是

= {AND , OR, NOT , IF}
= { AND，OR，NOT，IF }

taking 2, 2, 1, and 3 arguments, respectively.
分别采用 2,2,1 和 3 个参数。

The I F function is the commo n LISP function that per-forms the I F - T H E N - E L S E operation. That is, the I F func-
I f 函数是执行 i f-t h e n-e l s 操作的公共 n LISP 函数。也就是说，i f 函数

tion returns the results of evaluating its third argument (the 'else' clause) if its first argument is N I L (false) and other-wise returns the results o f evaluating its second argument (the 'then' clause). The above function set o~ is known to be sufficient to realize any Boolean function.

Tion 返回计算第三个参数(else 子句)的结果，如果它的第一个参数是 n i l (false)，而 other 返回计算第二个参数(then 子句)的结果。上面的函数集 o ~ 已知足以实现任何布尔函数。

Since genetic programming operates on an initial popu - lation o f randomly generated compositions of the available functions and terminals (and later performs genetic opera-tions, such as crossover, on these individuals), each primi-tive function in the function set should be well defined for any combination of arguments from the range of values returned by every primitive function that it may encounter and the value of every terminal that it may encounter. The above function set f f of primitive functions satisfies the closure property.

由于遗传编程是在可用函数和终端的随机组合的初始种群上进行操作(然后对这些个体进行遗传操作，例如交叉操作)，因此函数集中的每个基本函数都应该根据它可能遇到的每个基本函数返回的值的范围以及它可能遇到的每个终端的值，对参数的任何组合进行良好的定义。上面的函数集 f 满足闭包性质。

The search space for this problem is the set of all LISP S-expressions that can be recursively composed of the primitive functions from the function set f f and terminals from the terminal set f . Another way to look at the search space is that the Boolean multiplexer function with k § 2 k arguments is a particular one of 2 k+2k possible Boolean functions of k + 2 k arguments. F o r example,

该问题的搜索空间是所有 LISP s 表达式的集合，这些表达式可以由函数集 f 中的基元函数和终端集 f 中的终端递归组成。查看搜索空间的另一种方法是，带有 k2k 参数的布尔多路复用器函数是 k + 2k 参数的 2k + 2k 可能的布尔函数中的一个。例如,

when k = 3, then k + 2 k = 11 and this search space is o f size 2 211 . That is, the search space is of size 22~ which is a p p r o x i m a t e l y 10 616 .

当 k = 3 时，k + 2 k = 11 这个搜索空间是 o f 大小 2211。也就是说，搜索空间的大小是 22 ~ 也就是 p p r o x i m a t e y 10616。

The third major step in preparing to use genetic program - ming is the identification of the fitness measure for evaluat-ing the goodness of an individual S-expression in the population. Fitness is often evaluated over a number of fit-ness cases -- just as computer programs are typically debugged by examining their output over a number of test cases. The set of fitness cases must be representative of the problem as a whole. The reader may find it helpful to think of these fitness cases as the 'environment' in which the genetic population of computer programs must adapt. There are 211 = 2048 possible combinations of the 11 arguments *aoalazdodldzd3d4dsd6d7* along with the associated correct value of the 11-multiplexer function. F o r this particular problem, we use the entire set of 2048 combinations of arguments as the fitness cases for evaluat-ing fitness (although we could, of course, use sampling).

在准备使用遗传程序明的第三个主要步骤是确定适应性措施，以评估个体在群体中的 s 表达的好坏。适应性通常通过许多适应性案例进行评估——就像计算机程序通常通过检查它们在许多测试案例中的输出进行调试一样。适应性案例集必须代表整个问题。读者可能会发现，把这些适应性案例看作是计算机程序的遗传群体必须适应的“环境”是有帮助的。11 个参数 aoalazdodldzd3d4dsd6d7 的 211 = 2048 可能的组合，以及 11-multiplexer 函数的相关正确值。对于这个特殊的问题，我们使用整套 2048 个参数的组合作为适应性案例来评估适应性(当然，我们可以使用抽样)。

We begin by defining raw fitness in the simplest way that comes to mind using the natural terminology of the prob - lem. The raw fitness of a LISP S-expression in this problem is simply the number of fitness cases (taken over all 2048 fitness cases) where the Boolean value returned by the S-expression for a given combination of arguments is the correct Boolean value. Thus, the raw fitness of an S-expres-sion can range over 2049 different values between 0 and 2048. A raw fitness of 2048 denotes a 100% correct indivi-dual S-expression.

我们首先使用问题的自然术语以最简单的方式定义原始适应性。在这个问题中，LISP s 表达式的原始适应度只是适应度情况的个数(取自所有 2048 个适应度情况)，其中，对于给定的参数组合，s 表达式返回的布尔值就是正确的布尔值。因此，一个 s 表达式的原始适应度可以在 0 到 2048 之间的 2049 个不同的值范围内。2048 的原始适应性表示 100% 正确的个体双 s 表达式。

It is useful to define a fitness measure called standardized fitness where a smaller value is better and a zero value is best. Since a bigger value of raw fitness is better for this problem, standardized fitness is different from raw fitness

定义一种称为标准化适应性的适应性度量是有用的，在这种适应性度量中，较小的值更好，零值最好。由于原始适应度的大值对这个问题更有利，标准适应度与原始适应度是不同的

for this problem. In particular, standardized fitness equals the maximum possible value of raw fitness *rmax* (i.e. 2048) minus the observed raw fitness. The standardized fitness can also be viewed as the sum, taken over all 2048 fitness cases, of the Hamming distances (errors) between the Boolean value returned by the S-expression for a given combination of arguments and the correct Boolean value. The Hamming distance is zero if the Boolean value returned by the S-expression agrees with the correct Boolean value and is one if it disagrees. Thus, the sum of the Hamming distances is equivalent to the number of mismatches.

这个问题。特别地，标准化适应度等于原始适应度 rmax (即 2048)的最大可能值减去观察到的原始适应度。标准化适应度也可以被看作是 s 表达式返回的给定参数组合的布尔值与正确的布尔值之间的汉明距离(错误)的总和，取自所有 2048 个适应度情况。如果 s 表达式返回的布尔值与正确的布尔值一致，则汉明距离为零，如果不一致，则为 1。因此，汉明距离的和等于不匹配的次数。

The fourth major step in using genetic programming is selecting the values of certain parameters. The two major parameters that are used to control the process are the population size M and the maximum number of gen-erations Ngen to be run. Ngen is 51 throughout this article. Our choice of 4000 as the population size for this problem reflects an estimate on our part as to the likely complexity of this problem and the practical limitations of available computer memory.

使用遗传规划的第四个主要步骤是选择某些参数的值。用于控制这个过程的两个主要参数是种群大小 m 和将要运行的最大代数 Ngen。在本文中，Ngen 是 51。我们选择 4000 作为这个问题的人口规模，反映了我们对这个问题可能的复杂性和可用计算机内存的实际局限性的估计。

In addition, genetic programming is controlled by a number of additional secondary parameters. Our choice of values for the various secondary parameters that control the runs of genetic programming are the same default values as we have used on numerous other problems (Koza 1992a). Specifically, each new generation is created from the preceding generation by applying the fitness pro-portionate reproduction operation to 10% of the popu-lation and by applying the crossover operation to 90% of the population (with both parents selected with a prob - ability proportionate to fitness). In selecting crossover points, 90% were internal (function) points of the tree and 10% were external (terminal) points of the tree. F o r the practical reason o f avoiding the expenditure of large amounts of computer time on an occasional oversized pro-gram, the depth o f initial random programs was limited to 6 and the depth of programs created by crossover was limited to 17. The individuals in the initial rando m generation were generated so as to obtain a wide variety o f different sizes and shapes among the S-expressions. Fitness is 'adjusted' to emphasize small differences near zero. Spousal selection was also fitness proportionate . Details of the selection of these secondary parameters can be found in Koza (1992a). We believe that sufficient information is provided herein and in Koza (1992a) to allow replication of the experimental results reported herein, within the limits inherent in a probabilistic algorithm. C o m m o n LISP soft-ware for genetic programming is listed in Koz a (1992a).

此外，遗传编程由许多额外的次要参数控制。我们为控制遗传程序运行的各种次要参数选择的值与我们在许多其他问题上使用的默认值相同(Koza 1992a)。具体而言，每一个新一代都是通过对 10% 的人口应用适应度比例繁殖操作，以及对 90% 的人口应用交叉操作(父母双方的选择能力与适应度成比例的概率)从上一代中创造出来的。在选择交叉点时，90% 是树的内部(功能)点，10% 是树的外部(末端)点。由于避免在偶尔的超大程序上花费大量计算机时间的实际原因，初始随机程序的深度被限制为 6，交叉生成的程序的深度被限制为 17。在最初的随机 m 生成的个人，以获得各种不同的大小和形状的 s 表达式。适应性被“调整”以强调接近零的微小差异。配偶选择也是适合的比例。这些次要参数的选择细节可以在 Koza (1992a)中找到。我们相信，在本文和 Koza (1992a)中提供了足够的信息，以允许在概率算法固有的限度内重复本文所报告的实验结果。用于遗传编程的 LISP 软件在 Koz a (1992a)中列出。

Finally, the fifth major step in preparing to use genetic programming is the selection of the criterion for terminat-ing a run and the selection o f the method for designating a result. In this problem we have a way to recognize a solu-tion when we find it. When the raw fitness is 2048 (i.e. the standardized fitness is zero), we have a 100% correct

最后，准备使用遗传规划的第五个主要步骤是选择终止运行的标准和选择指定结果的方法。在这个问题中，当我们找到一个解决方案时，我们有一种识别它的方法。当原始适应度为 2048(即标准适应度为零)时，我们有 100% 正确

94

solution to this problem. Thus, we terminate a run after a specified maximum number of generations Ngen (e.g. 51) or earlier if we find an individual with a raw fitness of 2048. For all the problems in this article, we will terminate a given run after 51 generations and we designate the best single individual in the population at the time of ter-mination as the result of genetic programming.

解决这个问题的方法。因此，如果我们找到一个原始适应度为 2048 的个体，我们将在指定的最大代数 Ngen (例如 51)或更早的代数之后终止运行。对于本文中的所有问题，我们将在 51 代之后终止一个给定的传代，并且我们指定终止时种群中最好的个体作为遗传编程的结果。

We now illustrate genetic programming by discussing one particular run of the Boolean 11-multiplexer in detail. The process begins with the generation of the initial random population (i.e. generation 0).

我们现在通过详细讨论布尔 11 多路复用器的一个特定运行来说明遗传编程。这个过程开始于初始随机种群的生成(即第 0 代)。

Predictably, the initial random population includes a variety of highly unfit individuals. Man y individual S-expressions in this initial random population are merely constants, such as the contradictory (AND A0 (NOT A0)). Other individuals are passive and merely pass an input through as the output, such as (NOT (NOT A1)). Other individuals are inefficient, such as (OR 9 7 D7). Some of these initial random individuals base their deci-sion on precisely the wrong arguments, such as (IF DO A0 A2). This individual uses the data bit DO to decide what output to take. Many of the initial random indi-viduals are partially blind in that they do not incorporate all 11 arguments that are known to be necessary to solve the problem. Some S-expressions are just nonsense, such as

可以预见，初始随机种群包括各种高度不适合的个体。在这个初始随机种群中，Man y 个体的 s 表达式仅仅是常数，例如矛盾(AND A0(NOT A0))。其他个体是被动的，只是将输入作为输出传递，比如(NOT (NOT A1))。其他个体效率低下，如(OR 97 D7)。这些最初的随机个体中的一些，他们的决定完全基于错误的论点，例如(IF DO A0 A2)。这个个体使用数据位 DO 来决定输出什么。许多最初的随机个体是部分盲目的，因为他们没有包含所有 11 个已知的解决问题所必需的参数。一些 s 表达式只是胡说八道，比如

(IF (IF (IF 9 2 9 2 D2) D2 92) D2 D2).
(IF (IF (IF (IF 9292 D2) d292) D2 D2).

None the less, even in this highly unfit initial random population, some individuals are somewhat more fit than others. For this particular run, the individuals in the initial random population had values of standardized fitness rang-ing from 768 mismatches (i.e. 1280 matches) to 1280 mis-matches (i.e. 768 matches).

尽管如此，即使在这个非常不适合的初始随机群体中，有些个体还是比其他个体更适合。对于这个特定的运行，个人在最初的随机人口的标准化适应值范围从 768 错配(即 1280 个匹配)到 1280 错配(即 768 个匹配)。

The worst individual in the population for the initial random generation was

在最初的随机世代中，人口中最差的个体是

(OR (NOT A1) (NOT (IF (AND 1 2 10) 0 7   93))).
(或(非 A1)(非(IF (AND 1210)0793)))。

This individual had a standardized fitness of 1280 (i.e. raw fitness of only 768).

这个人的标准化适应度为 1280(即原始适应度仅为 768)。

As it happens, a total of 23 individuals out of the 4000 in this initial random population tied with the highest score of 1280 matches on generation 0. One of these 23 high-scoring individuals was the S-expression

碰巧的是，在这个最初的随机种群中，4000 个个体中有 23 个个体与第 0 代的 1280 个匹配的最高分并列。这 23 个高分个体中的一个是 s 表达式

(IF A0 D1 D2).
(IF A0 D1 D2).

This individual scores 1280 matches by scoring 512 matches for the one quarter (i.e. 512) of the 2048 fitness cases for which A2 and A1 are both N I L and by scoring an addi-tional 768 matches on 50% of the remaining three quarters (i.e. 1536) of the fitness cases.

在 2048 宗 A2 及 A1 均为 n i l 的个案中，该名人士在其中一个季度(即 512 个)取得 512 分，并在余下三个季度(即 1536 个)的 50% 的个案(即 1536 个)取得额外 768 分，为 1280 场比赛取得佳绩。

This individual has obvious shortcomings. Notably, it is partially blind in that it uses only 3 of the 11 necessary terminals of the problem. As a consequence of this fact alone, this individual cannot possibly be a correct solution to the problem. This individual none the less does some

这个人有明显的缺点。值得注意的是，它部分是盲目的，因为它只使用了问题的 11 个必要终端中的 3 个。由于这个事实本身的结果，这个个体不可能是问题的正确解决方案。尽管如此，这个人还是做了一些事情

things right. For example, it uses 1 of the 3 address bits (A0) as the basis for its action. It could easily have done this wrong and used 1 of the 8 data bits. In addition, this individual uses only data bits (D1 and D2) as its output. It could have done this wrong and used address bits. More-over, if A0 (which is the low-order binary bit of the 3-bit address) is T (True), this individual selects 1 of the 3 odd-numbered data bits (D1) as its output. Moreover, if A0 is NIL, this individual selects 1 of the 3 even-numbered data bits (D2) as its output. In other words, this individual correctly links the parity of the low-order address bit A0 with the parity of the data bit it selects as its output. This individual is far from perfect, but it is far from being with-out merit. It is more fit than 3977 of the 4000 individuals in the population.

事情是对的。例如，它使用 3 个地址位中的 1 个(A0)作为操作的基础。它很容易犯这个错误，使用了 8 个数据位中的 1 个。另外，这个人只使用数据位(D1 和 D2)作为输出。它可能做错了这一点，并使用了地址位。此外，如果 A0(3 位地址的低阶二进制位)为 t (True)，则此个体选择 3 个奇数编号数据位(D1)中的 1 个作为其输出。此外，如果 A0 是 NIL，这个个体选择 3 个偶数数据位(D2)中的 1 个作为它的输出。换句话说，这个个体正确地将低阶地址位 A0 的奇偶校验与它选择作为其输出的数据位的奇偶校验联系起来。这个个体远非完美，但也远非毫无价值。它比人口中 4000 个个体中的 3977 个更合适。

The average standardized fitness for all 4000 individuals in the population for generation 0 is 985.4. This value of average standardized fitness for the initial random popu-lation forms the baseline and serves as a useful benchmark for monitoring later improvements in the average standard-ized fitness of the population.

0 代人群中所有 4000 个个体的平均标准化适合度为 985.4。初始随机群体的平均标准化适应度值构成基线，并作为监测后来群体平均标准化适应度改善情况的有用基准。

The hits histogram is a useful monitoring tool based on the auxiliary hits measure. This histogram provides a way of viewing the population as a whole for a particular gen-eration. The horizontal axis of the hits histogram is the number of hits (i.e. matches, for this problem) and the vertical axis is the number of individuals in the population scoring that number of hits. Fifty different levels of fitness are represented in the hits histogram for the population at generation 0 of this problem. In order to make this histo-gram legible for this problem, we have divided the hori-zontal axis into buckets of size 64. For example, 1553 individuals out of 4000 (i.e. about 39%) had between 1152 and 1215 matches (hits). This well-populated range includes the mode of the distribution which occurs at 1152 matches (hits). There are 1490 individuals with 1152 matches (hits). Figure 6 shows the hits histogram of the population for generation 0 of this run of this problem.

命中直方图是基于辅助命中度量的有用监测工具。这种直方图提供了一种方法，可以将某一代人的人口作为一个整体来看待。命中直方图的水平轴是命中次数(即匹配，对于这个问题)，而垂直轴是人口中得分该命中次数的个人数。在这个问题的第 0 代人口的命中直方图中表示了 50 个不同的适应度水平。为了使这个直方图对于这个问题可读，我们将水平轴划分为 64 大小的桶。例如，4000 人中有 1553 人(约 39%)有 1152 到 1215 个匹配(点击率)。这个人口众多的范围包括发生在 1152 个匹配(点击)的分布模式。有 1490 个人有 1152 个匹配(点击率)。图 6 显示了这个问题的第 0 代人口的命中直方图。

The Darwinian reproduction operation and the genetic crossover operation are then applied to parents selected from the current population with probabilities proportion-ate to fitness to breed a new population. When these operations are completed, the new population (i.e. the new generation) replaces the old population.

然后将达尔文生殖操作和遗传交叉操作应用于从当前群体中选择的具有适应性比例的亲本进行繁殖。当这些操作完成后，新种群(即新一代)将取代旧种群。

The initial random generation is an exercise in blind random search. In going from generation 0 to generation

最初的随机一代是盲目随机搜索的一个练习。从第 0 代到第 0 代

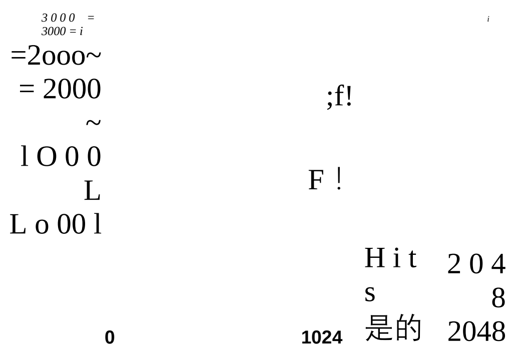

Fig. 6. *Hits histogram for generation 0*
图 6. 第 0 代的 Hits 直方图

1, genetic programming works with the inevitable genetic variation existing in an initial random population. The search is a parallel search of the search space because there are 4000 individual points involved.
1、遗传规划与存在于初始随机种群中的不可避免的遗传变异有关。搜索是对搜索空间的平行搜索，因为涉及到 4000 个单独的点。

Although the vast majority of the new offspring are again highly unfit, some of them tend to be somewhat more fit than others. Moreover, over a period of time and many gen-erations, some of them tend to be slightly more fit than those existing in the earlier generation. In this run, the aver-age standardized fitness of the population immediately begins improving (i.e. decreasing) from the baseline value of 985.4 for generation 0 to about 891.9 for generation 1. We typically see this kind of generally improving trend in average standardized fitness from generation to gen-eration. As it happens, in this particular run of this particu-lar problem, the average standardized fitness improves (i.e. decreases) monotonically between generation 2 and gen-eration 9 and assumes values of 845, 823, 763, 731, 651, 558, 459, and 382, respectively. We usually see a generally improving trend in average standardized fitness from gen-eration to generation, but not necessarily a monotonic improvement.
尽管绝大多数新的后代都非常不健康，但是他们中的一些人往往比其他人更健康。此外，经过一段时间和许多世代，他们中的一些人往往比那些存在于上一代的人更健康一些。在这种情况下，人口的平均标准化适应性立即开始提高(即下降)，从 0 代的基线值 985.4 提高到第 1 代的约 891.9。我们通常看到这种从一代到另一代的平均标准化适应性的普遍改善趋势。碰巧，在这个特殊问题的特殊运行中，平均标准化适应度在第 2 代和第 9 代之间单调地提高(即降低)，并且假设值分别为 845、823、763、731、651、558、459 和 382。我们通常看到从一代到另一代的平均标准化适应性总体上有改善的趋势，但不一定是单调的改善。

In addition, we similarly usually see a generally improv-ing trend in the standardized fitness of the best single indi-vidual in the population from generation to generation. As it happens, in this particular run of this particular problem, the standardized fitness of the best single individual in the population improves (i.e. decreases) monotonically between generation 2 and generation 9. In particular, it assumes values of 640, 576, 384, 384, 256, 256, 128, and 0 (i.e. a perfect score), respectively.
此外，我们同样经常看到，人口中最优秀的单个个体的标准化适应性从一代到另一代普遍呈现出改善的趋势。碰巧的是，在这个特殊问题的特殊运行过程中，种群中最好的单个个体的标准化适应性在第 2 代和第 9 代之间单调地提高(即降低)。特别是，它假定值分别为 640、576、384、384、256、256、128 和 0(即完美分数)。

On the other hand, the standardized fitness of the worst single individual in the population fluctuates consider-ably. For this particular run, the standardized fitness of the worst individual starts at 1280, fluctuates considerably between generations 1 and 9, and then deteriorates (increases) to 1792 by generation 9.
另一方面，人口中最差的个体的标准化适应性可以考虑波动。对于这个特定的运行，标准化的健康状况最差的个体从 1280 年开始，在第 1 代和第 9 代之间波动很大，然后恶化(增加)到 1792 年的第 9 代。

Figure 7 shows the standardized fitness (i.e. mismatches) for generations 0 through 9 of this run for the best
图 7 显示了这次运行的第 0 代到第 9 代的最佳标准化适应性(即错配)

Fig. 7. *Standardized fitness of worst-of-generation individual, aver-age standardized fitness of population, and standardized fitness of best-of-generation individualfor generations 0 through 9*
图 7。最差世代个体的标准化适应度，人口的平均标准化适应度，以及 0—9 代最佳世代个体的标准化适应度

single individual in the population, the worst single individual in the population, and the average for the population.
人口中的单个个体，人口中最差的单个个体，以及人口的平均数。

In generation 1, the raw fitness for the best single individual in the population rises to 1408 matches (i.e. standardized fitness of 640). Only one individual in the population attained this high score of 1408 in generation 1, namely
在第一代，人口中最好的单个个体的原始适应度上升到 1408 匹配(即标准化适应度为 640)。人口中只有一个个体在第一代达到了 1408 的高分，即

(IF A0 (IF A2 D7 D3) DO).
(IF A0(IF A2 D7 D3) DO).

Note that this individual performs better than the best individual from generation 0 for two reasons. First, this individual considers two of the three address bits (A0 and A2) in deciding which data bit to choose as output, whereas the best individual in generation 0 considered only 1 of the 3 address bits (A0). Second, this best indi-vidual from generation 1 incorporates 3 of the 8 data bits as its output, whereas the best individual in generation 0 incorporated only 2 of the 8 potential data bits as output. Although still far from perfect, the best individual from generation 1 is less blind and more complex than the best individual of the previous generation. This best-of-generation individual consists of 7 points, whereas the best-of-generation individual from generation 0 consisted of only 4 points. Note that these 21 individuals are not just copies of the best-of-generation individual from generation 1. Instead, they represent a number of different programs with the same fitness, but different structure and behaviour.
请注意，由于两个原因，此个体的性能优于第 0 代中的最佳个体。首先，这个个体在决定选择哪个数据位作为输出时考虑三个地址位中的两个(A0 和 A2)，而第 0 代中最好的个体只考虑三个地址位中的一个(A0)。其次，第 1 代的最佳个体将 8 个数据位中的 3 个作为输出，而第 0 代的最佳个体仅将 8 个潜在数据位中的 2 个作为输出。尽管还远远不够完美，但第一代的最佳个体比上一代的最佳个体更少盲目，更复杂。这个最好的一代个体由 7 分组成，而来自第 0 代的最好的一代个体只有 4 分。请注意，这 21 个人不仅仅是第 1 代中最好的一代人的复制品。相反，他们代表了许多不同的程序，具有相同的适应性，但是不同的结构和行为。

In generation 2, the best raw fitness remained at 1408; however, the number of individuals in the population sharing this high score rose from 1 to 21. The high point of the hits histogram advanced from 1152 for generation 0 to 1280 for generation 2. There are 1620 individuals with 1280 hits.
在第二代中，最好的原始适应性保持在 1408；然而，在群体中分享这一高分的个体数量从 1 增加到 21。点击直方图的高点从第 0 代的 1152 个上升到第 2 代的 1280 个。有 1620 个人有 1280 个点击。

In generation 3, one individual in the population attained a new high score of 1472 matches (i.e. standardized fitness of 576). This individual has 16 points and is
在第三代，人口中的一个个体获得了 1472 个匹配的新高分数(即标准化适应度为 576)。这个人有 16 分，是

(IF A2 (IF A0 D7 D4)
(IF A2(IF A0 D7 D4)

(AND (IF (IF A2 (NOT D5) A0) D3 D2) D2)).
(及(IF (IF A2(NOT D5) A0) D3 D2) D2))。

Generation 3 shows further advances in fitness for the population as a whole. The number of individuals with 1280 hits (the high point for generation 2) has risen to 2158 for generation 3. Moreover, the centre of gravity of the fitness histogram has shifted significantly from left to right. In particular, the number of individuals with 1280 hits or better has risen from 1679 in generation 2 to 2719 in generation 3.
第三代显示了整个人口适应性的进一步提高。1280 次点击的个体数量(第二代的最高点)已经上升到第三代的 2158 个。此外，健身直方图的重心已经从左向右显着转移。特别是，拥有 1280 个或更多点击的个体数量已经从第二代的 1679 个上升到第三代的 2719 个。

In generations 4 and 5, the best single individual has 1664 hits. This score is attained by only one individual in generation 4, but by 13 individuals in generation 5. One of these 13
在第 4 代和第 5 代中，最优秀的个体有 1664 个点击率。这个分数仅由第 4 代的一个个体获得，但由第 5 代的 13 个个体获得。其中之一

individuals is
个别人士

```
      (IF A0 (IF A2 D7  D3)
      (IF A0(IF A2 D7 D3)

         (IF A2 D4 (IF A1 D2 (IF A2 D7 DO)))).
         (IF A2 D4(IF A1 D2(IF A2 D7 DO))).
```

Note that this individual uses all three address bits (A2, A1, and A0) in deciding upon the output. It also uses 5 of the 8 data bits. By generation 4, the high point of the histogram has moved to 1408 with 1559 individuals. In generation 6, 4 individuals attain a score of 1792 hits. The high point of the histogram has moved to 1536 hits. In generation 7, 70 individuals attain this score of 1792 hits.
注意，这个人使用所有三个地址位(A2，A1 和 A0)来决定输出。它还使用了 8 个数据位中的 5 个。到了第四代，直方图的最高点已经移到了 1408，有 1559 个个体。在第 6 代中，4 个个体获得了 1792 次命中的得分。直方图的最高点移动到了 1536 次点击。在第 7 代中，70 个个体获得了 1792 次点击的得分。

In generation 8, there are four best-of-generation individuals. They all attain a score of 1920 hits. The mode (high point) of the histogram has moved to 1664, and 1672 individuals share this value. Moreover, an additional 887 individuals score 1792.

在第八代中，有四个最优秀的个体。他们都获得了 1920 年的点击率。直方图的模式(高点)已经移动到 1664，并且 1672 个人分享这个值。此外，另有 887 人得分为 1792。

In generation 9, one individual emerges with a 100% perfect score of 2048 hits. That individual is

在第 9 代中，有一个个体在 2048 年的点击率中得到了 100% 的满分

```
      (IF A0 (IF A2 (IF A1 D7 (IF A0 D5 DO))
      (IF A0(IF A2(IF A1 D7(IF A0 D5 DO)))

         (IF A0 (IF A1 (IF A2 D7 D3) D1)
         (如果 A0(如果 A1(如果 A2 D7 D3) D1)

            DO))
            DO)

         (IF A2 (IF A1 D6 D4)
         (IF A2(IF A1 D6 D4)

            (IF A2 D4
            (如果 A2 D4

                (IF A1 D2 (IF A2 D7 DO)))))
              (如果 A1 D2(如果 A2 D7 DO)))
```

Figure 8 shows the 100% correct individual from gen-eration 9. This 100% correct individual from generation 9 is a hierarchical structure consisting of 37 points (i.e. 12 functions and 25 terminals).

图 8 显示了来自第 9 代的 100% 正确的个体。第 9 代的这个 100% 正确的个体是一个由 37 个点(即 12 个函数和 25 个终端)组成的层次结构。

Note that the size and shape of this solution emerged from genetic programming. This particular size and this particular hierarchical structure was not specified in advance. Instead, it evolved as a result of reproduction, crossover, and the relentless pressure of fitness. In gen-eration 0, the best single individual in the population had 12 points. The number of points in the best single indi-vidual in the population varied from generation to gen-eration. It was 4 in generation 0, while it was 37 for generation 9.

请注意，这个解决方案的大小和形状来自遗传编程。这个特殊的大小和这个特殊的层次结构没有事先指定。相反，它是由繁殖、交叉和无情的适应压力而演变而来的。在第 0 代，种群中最优秀的个体有 12 个点。人口中最好的单个个体的分数因代而异。第 0 代为 4，第 9 代为 37。

Fig. 8. *100% correct individual from generation 9*
图 8.100% 正确的第 9 代个体

K o z a

Generation 3
第三代

3000,
3000,
2000~
= 2000 ~

10002

Hits
0 1024 找到了 2048

Generation 5
第五代

3000-
3000-
2000;
2000;
1000 i
1000 i

I

. _
~ l

0 " 9 . .
i . . . =
0 英尺 9
英寸

译
者
注:

- 我-我。. . .

E l
E l

Hits
0 1024 找到了 2048

Generation 7
第七代

3000-
3000-
~
=
~ 2000~
= 2000 ~
1000 ~
1000 ~
0''' *,,,,,,,,,,,,,*,
0’’’ * ... ... * ,

Hits
0 1024 找到了 2048

Generation 9
第九代

3000-
3000-

2000"

2000」 !

1000~
g 1000 ~

, , .
i ; !
...
我...
!

0 ' . . . .
0, , , , ,.

Hits
0 1024 找到了 2048

Fig. 9. *Hits h&tograms for generations 3, 5, 7, and 9 for the 11-multiplexer*
图 9.11 多路复用器第 3、5、7 和 9 代的命中 h & tograms

This 100% correct individual can be simplified to
这 100% 正确的个人可以简化为

(IF A0 (IF A2 (IF A1 D7 D5) (IF A1 D3 D1))
(IF A0(IF A2(IF A1 D7 D5)(IF A1 D3 D1))

(IF A2 (IF A1 D6 D4) (IF A1 D2 DO))).
(如果 A1 D6 D4)(如果 A1 D2 DO))。

When so rewritten, it can be seen that this individual correctly performs the l 1-multiplexer function by first examining address bits A0, A2, and A1 and then choosing the appropriate 1 of the 8 possible data bits.
当这样重写时，可以看到这个个体通过首先检查地址位 A0、 A2 和 A1，然后从 8 个可能的数据位中选择适当的 1，正确地执行了 l1 多路复用器功能。

Figure 9 shows the hits histograms for generations 3, 5, 7, and 9 of this run. As one progresses from generation to gen-eration, note the left-to-right 'slinky' undulating movement of the centre of mass of the histogram and the high point of the histogram. This movement reflects the improvement of the population as a whole as well as the best single indi-vidual in the population. There is a single 100% correct individual with 2048 hits at generation 9; however, because of the scale of the vertical axis of this histogram, it is not visible in a population of size 4000.
图 9 显示了该运行的第 3、5、7 和 9 代的命中直方图。随着一代又一代的进步，请注意直方图的质量中心和直方图的高点从左到右的“弹性”起伏运动。这种运动反映了整个人口的改善以及人口中最好的单个个体。有一个 100% 正确的个体，在第 9 代有 2048 个点击; 然而，由于这个直方图的垂直轴的规模，它是不可见的，在人口大小 4000。

Further insight can be gained by studying the genea-logical audit trail consisting of a complete record of the details of each genetic operation that is performed at each generation. The creative role of crossover and case-splitting is illustrated by an examination of the genealogical audit trail for the 100% correct individual emerging at gen-eration 9.
进一步的洞察力可以通过研究遗传逻辑审计线索获得，该线索包括每一代遗传操作的完整细节记录。通过对第 9 代出现的 100% 正确的个体的系谱审计线索的检查，说明了交叉和案例分割的创造性作用。

The 100% correct individual emerging at generation 9 is the child resulting from the most common genetic operation
在第 9 代出现的 100% 正确的个体是由最常见的遗传操作产生的孩子

Fig. 10. *First parent (scoring 1792 hits)from generation 8for 100% correct individual in generation 9*
图 10. 第 8 代的第一个父母(得分 1792 次命中)在第 9 代中 100% 正确的个体

used in the process, namely crossover. The first parent from generation 8 had rank location of 58 in the population (with a rank of 0 being the very best) and scored 1792 hits (out of 2048). The second parent from generation 8 had rank location 1 and scored 1920 hits. Note that it is entirely typical that the individuals selected to participate in cross-over have relatively high rank locations in the population since crossover is performed among individuals in a mating pool created proportional to fitness.
在过程中使用，即交叉。第 8 代的第一个父母在人口中的排名位置为 58(排名为 0 是最好的)，得分为 1792(2048 年)。第 8 代的第二个父母有排名位置 1，得到 1920 个点击。请注意，这是完全典型的，个人选择参加交叉有相对较高的等级位置在人口中，因为交叉是在个人之间进行的交配池创建成比例的适应性。

The first parent from generation 8 (scoring 1792) was
第 8 代的第一个父母(得分 1792)是

(IF A0 (IF a 2 D7 D3)
(IF A0(IF a2 D7 D3)

(IF A2 (IF A1 D6 D4)
(IF A2(IF A1 D6 D4)

(IF A2 D4
(如果 A2 D4

(IF A1 D2 (IF A2 D7 DO))))).
(如果 A1 D2(如果 A2 D7 DO)))。

Figure 10 shows this first parent from generation 8. Note that this first parent starts by examining address bit A0. If A0 is T,

expression by removing the two extraneous tests and
通过删除两个无关的测试和

removing the D7 (which is unreachable). This sub-
移除 D7(无法访问)

expression simplifies to the following:
表达式简化如下:

(IF A2 (IF A1 D6 D4)
(IF A2(IF A1 D6 D4)

(IF a l D2 DO))
(如属 ld2 的话)

the underlined portion then examines address bit A2. It then, partially blindly, makes the output equal D7 or D3 without even considering address bit A1. More-over, the underlined portion of this individual does not even contain data bits D1 and D5.
图 10 显示了第 8 代的第一个父代。注意，这个第一个父类从检查地址位 A0 开始。如果 A0 是 t，带下划线的部分则检查地址位 A2。然后，它部分地盲目地使输出等于 D7 或 D3，甚至没有考虑地址位 A1。更重要的是，这个单独的划线部分甚至不包含数据位 D1 和 D5。

On the other hand, when A0 is NIL, this first parent is 100% correct. In that event, it examines A2 and, if A2 is T, it then examines A1 and makes the output equal to D6 or D4 according to whether A1 is T or NIL. Moreover, if A2 is NIL, it twice retests A2 (unnecessarily, but harm-lessly) and then correctly makes the output equal to (IF A1 D2 DO). Note that the 100% correct portion of this first parent, namely, the sub-expression

另一方面，当 A0 为 NIL 时，第一个父类是 100% 正确的。在这种情况下，它检查 A2，如果 A2 是 t，它就检查 A1，并根据 A1 是 t 还是 NIL 使输出等于 D6 或 D4。此外，如果 A2 是 NIL，它会两次重新测试 A2(不必要，但无害)，然后正确地使输出等于(IF a1d2do)。注意第一个父表达式的 100% 正确部分，即子表达式

(IF A2 (IF A1 D6 D4)
(IF A2(IF A1 D6 D4)

(IF A2 D4 (IF A1 D2 (IF A2 D7 DO))))
(IF A2 D4(IF A1 D2(IF A2 D7 DO)))

is itself a 6-multiplexer. This embedded 6-multiplexer tests A2 and A1 and correctly selects amongst D6, D4, D2, and DO. This fact becomes clearer if we simplify this sub-
本身就是一台 6 路复用器。这个嵌入式 6 路复用器测试 A2 和 A1，并在 D6，D4，D2 和 DO 中正确选择。如果我们简化这个子代码，这个事实就会变得更加清晰

In other words, this imperfect first parent handles part of its environment correctly and part of its environment incorrectly. In particular, this first parent handles the even-numbered data bits correctly and is partially correct in handling the odd-numbered data bits.

换句话说，这个不完美的第一个父类正确地处理了它的部分环境，也错误地处理了它的部分环境。特别是，这个第一父类正确地处理偶数数据位，并且在处理奇数数据位时部分正确。

The tree representing this first parent has 22 points. The crossover point chosen at random at the end of generation 8 was point 3 and corresponds to the second occurrence of the function IF. That is, the crossover fragment consists of the incorrect underlined sub-expression

表示第一个父级的树有 22 个点。在第 8 代结束时随机选择的交叉点是点 3，对应于函数 IF 的第二次出现。也就是说，交叉片段由不正确的下划线子表达式组成

```
                    (IF A2 D7 D3).
                    (IF A2 D7 D3).
```

The second parent from generation 8 (scoring 1920 hits)
第八代的第二个父母(得分 1920 点击量)

w a s
W a s

```
         (IF A0 (IF A0
         (如果 A0(如果 A0

           (IF A2 (IF A1 D7 (IF A0 D5 DO))
           (IF A2(IF A1 D7(IF A0 D5 DO))

             (IF A0 (IF a l       (IF A2 D7
               (IF A0(IF a1(IF A2 D7

                          D3)
                          D3)

                        D1)
                        D1)

                      DO))
                      DO)

             (IF A1 D6 D4))
             (IF A1 D6 D4)

           (IF A2 D4
           (如果 A2 D4

             (IF A1 D2
             (如果 a1d2

               (IF A0 D7 (IF A2 D4 DO)))))
               (如果 A0 D7(如果 A2 D4 DO)))
```

Figure 11 shows the second parent from generation 8.
图 11 显示了第 8 代的第二个父代。

Fig. 11. *Second parent (scoring 1920 hits)from generation 8 for 100% correct individual in generation 9*
图 11. 第 8 代的第二个父母(得分 1920 次命中)在第 9 代中 100% 正确的个体

98 Koza

iplexer fr(xn"~
Ise~and~ltparent
using I A 1 to select]
Iplexer fr (xn" ~ Ise ~ and
~ ltparent using i a1 to
select]
amongst D7, D5,]
在 D7，D5 之间]
D3 and D1 J
D3 和 D1 j

f6-Multiplexer from"~
/ first parent using /
IA2 and A 1 to select I
F6- 来自"/first parent
使用/IA2 和 a1 选择 i
的多路复用器
/ amongst D6, D4, /
在 D6，D4,/之间
~ D2 and DO „1
~ D2 和 DO，1

Fig. 12. *Simplified 100% correct individual from generation 9 shown as a hierarchy o f two 6-multiplexers*
图 12。简化的 100% 正确的个体从第 9 代显示为两个 6-多路复用器的层次结构

The tree representing this second parent has 40 points. The crossover point chosen at random for this second parent was point 5. This point corresponds to the third occurrence of the function IF. That is, the crossover fragment consists of the underlined sub-expression of this second parent.
表示第二个父类的树有 40 个点。为第二个父节点随机选择的交叉点是点 5。这个点对应于函数 IF 的第三次出现。也就是说，交叉片段由第二个父节点的带下划线的子表达式组成。

This sub-expression of this second parent 100% correctly handles the case when A0 is T (i.e. the odd-numbered addresses). This sub-expression makes the output equal to D7 when the address bits are 111, to D5 when the address bits are 101, to D3 when the address bits are 011; and to D 1 when the address bits are 001.
这个第二个父表达式的子表达式 100% 正确地处理了 A0 为 t (即奇数地址)时的情况。这个子表达式使得当地址位为 111 时输出等于 D7，当地址位为 101 时输出等于 D5，当地址位为 011 时输出等于 D3，当地址位为 001 时输出等于 d1。

Note that the 100% correct portion of this second parent, namely the sub-expression
注意第二个父表达式的 100% 正确部分，即子表达式

(IF A2 (IF A1 D7 (IF A0 D5 DO))
(IF A2(IF A1 D7(IF A0 D5 DO))

(IF A0 (IF A1 (IF A2 D7 D3) D1) DO))
(如果 A0(如果 A1(如果 A2 D7 D3) D1) DO))

is itself a 6-multiplexer. This embedded 6-multiplexer in the second parent tests A2 and A1 and correctly selects amongst D7, D5, D3, and D1 (i.e. the odd-numbered data bits). This fact becomes clearer if we simplify this sub-expression of this second parent to the following:

fect parents contain complementary portions which, when mated together, produce a 100% correct offspring indi-vidual. In effect, the creative effect of the crossover operation blends the two cases of the implicitly 'case-split' environment into a single 100% correct solution.
完美的双亲包含互补的部分，当交配在一起时，产生 100% 正确的后代个体。实际上，交叉操作的创造性效果将隐式"案例分割"环境的两种情况融合到一个 100% 正确的解决方案中。

Figure 12 shows this case splitting by showing the 100% correct offspring from generation 9 as two 6-multiplexers: Fig. 13 also shows this simplified version of the 100% correct individual from generation 9.
图 12 通过将第 9 代 100% 正确的后代显示为两个 6- 多路复用器，显示了这种情况的分裂: 图 13 还显示了第 9 代 100% 正确个体的简化版本。

本身就是一台 6 路复用器。这个嵌入在第二个父系中的 6-multiplexer 测试 A2 和 A1，并在 D7、 D5、 D3 和 D1(即奇数数据位)之间正确选择。如果我们将这个第二个父类的子表达式简化为以下内容，这个事实就会变得更加清晰:

(IF A2 (IF A1 D7 D5)
(IF A2(IF A1 D7 D5)

(IF A1 D3 D1))
(IF A1 D3 D1)

In other words, this imperfect second parent handles part of its environment correctly and part of its environment incorrectly. This second parent does not do very well when A0 is NIL (i.e. the even-numbered data bits). This second parent correctly handles the odd-numbered data bits and incorrectly handles the even-numbered data bits.
换句话说，这个不完美的第二个父类正确地处理了它的部分环境，也错误地处理了它的部分环境。当 A0 是 NIL (即偶数数据位)时，这个第二个父类做得不是很好。第二个父类正确地处理奇数数据位，而错误地处理偶数数据位。

Even though neither parent is perfect, these two imper-
即使父母双方都不完美，这两个人也不完美

Fig. 13. *Simplified 100% correct individual from generation 9 shown as a hierarchy o f two 6-multiplexers*
图 13。简化的 100% 正确的个体从第 9 代显示为两个 6-multiplexer 的层次结构

Of course, not all crossovers between individuals are useful and productive. In fact, a large number of the indi-viduals produced by the genetic operations are useless. But the existence of a population of alternative solutions to a problem provides the ingredients with which genetic recombination (crossover) can produce some improved individuals. The relentless pressure of natural selection based on fitness then causes these improved individuals to be preserved and to proliferate. Moreover, genetic vari-ation and the existence of a population of alternative solutions to a problem make it unlikely that the entire population will become trapped on local maxima.

Interestingly, the same crossover that produced the 100% correct individual also produced a runt scoring only 256 hits. In this particular crossover, the two cross-over fragments not used in the 100% correct individual combined to produce an unusually unfit individual. This is one of the reasons why there is considerable variability from generation to generation in the worst single indi-vidual in the population.

As one traces the ancestry of the 100% correct individual created in generation 9 deeper back into the genealogical audit tree (i.e. towards earlier generations), one encounters parents scoring generally fewer and fewer hits. That is, one encounters more S-expressions that perform irrelevant, counterproductive, partially blind, and incorrect work. But if we look at the sequence of hits in the forward direc-tion, we see localized hill-climbing in the search space occurring in parallel throughout the population as the creative operation of crossover recombines complemen-tary, co-adapted portions of parents to produce improved offspring.

当一个人追踪第 9 代创建的 100% 正确个体的祖先回到家谱审计树的更深处(即朝向更早的几代)，一个人会遇到父母得分一般越来越少的点击。也就是说，一个人会遇到更多的 s 表达式，这些表达式执行的是不相关的、适得其反的、部分盲目的和不正确的工作。但如果我们观察前进方向的命中次序，我们会发现在搜索空间中，局部的爬山行为在整个种群中平行发生，因为交叉的创造性操作重新组合了父母互补的、共同适应的部分，以产生更好的后代。

The solution to the 11-multiplexer problem in this run was a hierarchy consisting of two 6-multiplexers. In a run where we applied genetic programming to the simpler Boolean 6-multiplexer, we obtained the following 100% correct solution:

解决 11 个多路复用器问题的方法是一个由两个 6 个多路复用器组成的层次结构。在一次运行中，我们将遗传编程应用于更简单的布尔 6-multiplexer，我们得到了以下 100% 正确的解决方案:

(IF (AND A0 A1) D3 (IF A0 D1 (IF A1 D2 DO))).

(IF (AND A0 A1) D3(IF A0 D1(IF A1 D2 DO))).

This solution to the 6-multiplexer is also a hierarchy. It is a hierarchy that correctly handles the particular fitness cases where (AND A0 A1) is true and then correctly handles the remaining cases where (AND A0 A1) is false.

6-multiplexer 的这个解决方案也是一个层次结构。它是一个层次结构，可以正确处理特定的适应性情况，其中 (AND A0 A1)为 true，然后正确处理剩余的情况，其中 (AND A0 A1)为 false。

Default hierarchies often emerge from genetic program-ming. A default hierarchy incorporates partially correct sub-rules into a perfect overall procedure by allowing the

默认层次结构常常出现在遗传程序开发中。默认层次结构将部分正确的子规则整合到一个完美的整体过程中

partially correct (default) sub-rules to handle the majority of the environment and by then dealing in a different way with certain specific exceptional cases in the environment. The S-expression above is also a default hierarchy in which the output defaults to
部分正确(默认)子规则来处理大部分环境，然后以不同的方式处理环境中的特定例外情况。上面的 s 表达式也是一个默认的层次结构，其输出默认为

(IF A0 D1 (IF A1 D2 DO))
(IF A0 D1(IF A1 D2 DO))

three quarters of the time. However, in the specific exceptional fitness case where both address bits (A0 and A1) are both T, the output is the data bit D3.
四分之三的时间。然而，在地址位(A0 和 A1)都是 t 的特定异常适应性情况下，输出是数据位 D3。

Default hierarchies are considered desirable in induction problems (Holland, 1986, Holland *et al.*, 1986, Wilson, 1988) because they are often parsimonious and they are a human-like way of dealing with situations.
默认层次结构在归纳问题中被认为是可取的(Holland，1986，Holland 等，1986，Wilson，1988)，因为它们通常是简约的，而且它们是一种类似于人类处理情况的方式。

5. Symbolic regression-empirical data
5. 符号回归-经验数据

An important problem area in virtually every area of science is finding the relationship underlying empirically observed values of the variables measuring a system. In practice, the observed data may be noisy and there may be no known way to express the relationships involved in a precise way.
几乎每个科学领域都有一个重要的问题，那就是找到衡量一个系统的变量的经验观测值之间的关系。在实践中，观察到的数据可能是嘈杂的，并且可能没有已知的方法来精确地表达所涉及的关系。

The learning of the Boolean multiplexer function is an example of the general problem of symbolic function identification (symbolic regression). In this section, we discuss symbolic regression as applied to real-valued functions over real-valued domains.
布尔复用器函数的学习是符号函数识别(符号回归)一般问题的一个例子。在本节中，我们讨论符号回归应用于实值域上的实值函数。

In conventional linear regression, one is given a set of values of various independent variable(s) and the corresponding values for the dependent variable(s). The goal is to discover a set of numerical coefficients for a linear combination of the independent variable(s) which minimizes some measure of error (such as the square root of the sum of the squares of the differences) between the given values and computed values of the dependent variable(s). Similarly, in quadratic regression, the goal is to discover a set of numerical coefficients for a quadratic expression which similarly minimizes error.
在传统的线性回归中，给出了各个自变量的一组值以及相应的自变量的值。目标是发现一组数值系数，用于自变量的线性组合，从而最小化给定值与因变量计算值之间的误差度量(如差值的平方和的平方根)。同样，在二次回归中，目标是发现一组二次表达式的数值系数，同样可以最小化误差。

Of course, it is left to the researcher to decide whether to do a linear regression, a quadratic regression, or a higher-order polynomial regression, or whether to try to fit the data points to some non-polynomial family of functions (e.g. sines and cosines of various periodicities, etc.). But, often, *the* issue is deciding what type of function most appropriately fits the data, not merely computing the numerical coefficients after the type of function for the model has already been chosen. In other words, the real problem is often *both* the discovery of the correct func-tional form that fits the data *and* the discovery of the appro-priate numeric coefficients that go with that functional form. We call the problem of finding, in symbolic form, a function that fits a given finite sample of data, by the name *symbolic regression.* It is 'data to function' regression.
当然，是否进行线性回归、二次回归或高阶多项式回归，或者是否试图将数据点拟合到某些非多项式函数族(如各种周期的正弦和余弦等)，都是由研究人员决定的。但是，通常的问题是决定什么类型的函数最适合数据，而不仅仅是在已经选择了模型的函数类型之后计算数值系数。换句话说，真正的问题往往是既要发现适合数据的正确函数形式，又要发现适合该函数形式的数值系数。我们把以符号形式找到一个适合给定有限数据样本的函数的问题称为符号回归。这就是“数据到函数”回归。

The problem of discovering empirical relationships from actual observed data is illustrated by the well-known non-linear econometric exchange equation

从实际观测数据中发现经验关系的问题可以用著名的非线性计量经济学交换方程来解释

M V
p -
P-m v
Q

This equation states the relationship between the gross national product Q of an economy, the price level P, the money supply M, and the velocity of money V.

这个方程描述了一个经济体的国民生产总值 q，物价水平 p，货币供应 m 和货币流通速度 v 之间的关系。

Suppose that our goal is to find the econometric model expressing the relationship between quarterly values of the price level P and the quarterly values of the three other quantities appearing in the equation. That is, our goal is to rediscover the relationship

假设我们的目标是找到表示价格水平 p 的季度价值与方程中其他三个数量的季度价值之间关系的计量经济学模型。也就是说，我们的目标是重新发现这种关系

M V
p -
P-m v
Q

from the actual observed noisy time series data. Moreover, suppose that certain additional economic data are also available which are irrelevant to this relationship, but not pre-identified as being irrelevant. Many economists believe that inflation (which is the change in the price level) can be controlled by the central bank via adjustments in the money supply M. Specifically, the 'correct' exchange equation for the United States in the postwar period is the non-linear relationship

从实际观测到的噪声时间序列数据。此外，假设某些额外的经济数据也是可用的，这些数据与这种关系无关，但没有事先确定为无关。许多经济学家认为，通货膨胀(物价水平的变化)可以通过调整货币供应量来控制。具体来说，战后美国的"正确"汇率方程是非线性关系

GD - (1.6527 * M2)
GD-(1.6527 * M2)
GNP82
GNP82

where 1.6527 is the actual long-term historic postwar value of the M2 velocity of money in the United States (Hallman *et al.* 1989). Interest rates are not a relevant variable in this well-known relationship.

其中 1.6527 是战后美国 M2 货币流通速度的实际长期历史价值(Hallman et al. 1989)。在这个众所周知的关系中，利率并不是一个相关的变量。

In particular, suppose we are given the 120 actual quarterly values from 1959:1 (i.e. the first quarter of 1959) to 1988:4 of the following four econometric time series.

特别是，假设我们得到以下四个计量经济学时间序列中从 1959:1(即 1959 年第一季度)到 1988:4 的 120 个实际季度值。

9 Inflation or price level P (the dependent variable here) is represented by the Gross National Product Deflator (normalized to 1.0) for 1982 (conventionally called GD).
9 通货膨胀或价格水平 p (这里的因变量)由 1982 年国民生产总值缩减指数(归一化为 1.0)(通常称为 GD)表示。

9 The gross national product of the economy Q (one of the independent variables) is represented by the annual rate for the United States Gross National Pro-duct in billions of 1982 dollars (conventionally called GNP82).
9 经济的国民生产总值 q (自变量之一)用美国 1982 年国民生产总值的年率(以十亿美元计)(通常称为 GNP82)表示。

9 The money supply M (another of the independent variables) is represented by the monthly values of the seasonally adjusted money stock M2 in billions of dollars, averaged for each quarter (conventionally called M2).
9 货币供应 m (另一个自变量)由经季节性调整的货币存量 M2 的月度值(以十亿美元计)表示，每个季度的平均值(通常称为 M2)。

9 Interest rates (an independent variable that happens to be irrelevant to the calculation here) are represented by the monthly interest rate yields of 3-month Treasury bills, averaged for each quarter (conventionally called FYGM3).
利率(一个与本文计算无关的独立变量)由 3 个月期美国国债的月利率收益率表示，每个季度的平均利率(通常称为 FYGM3)。

The four time series used here were obtained from the CITI-BASE data base of machine-readable econometric time series (Citibank, 1989).
这里使用的四个时间序列是从机器可读计量经济时间序列的 CITI-BASE 数据库中获得的(Citibank，1989)。

As a point of reference, the sum of the squared errors between the actual gross national product deflator G D from 1959:1 to 1988:4 and the fitted G D series calculated from the above model over the entire 30-year period involv-ing 120 quarters (1959:1 to 1988:4) is very small, namely 0.077193. The correlation R 2 was 0.993320.

作为一个参照点，1959:1 至 1988:4 期间的实际国民生产总值减缩指数 g d 与根据上述模型计算的整个 30 年期间 120 个季度(1959:1 至 1988:4)的拟合 g d 系列之间的平方误差之和非常小，即 0.077193。相关系数 r2 为 0.993320。

These 120 combinations of the above three independent variables (M2), and the associated value of the dependent variables (GD, GNP82, and FYGM3) are the set from which we will draw the fitness cases that will be used to evaluate the fitness of any proposed S-expression.

上述三个自变量(M2)和因变量(GD、 GNP82 和 FYGM3)的关联值的 120 个组合是我们将从中得出适应性案例的集合，这些适应性案例将用于评估任何提出的 s 表达式的适应性。

The goal is to find a function, in symbolic form, that is a good fit or perfect fit to the numerical data points. The solu-tion to this problem of finding a function in symbolic form that fits a given sample of data can be viewed as a search for
目标是找到一个符号形式的函数，这是一个很好的适合或完美的适合数字数据点。这个问题的解决方案是找到一个符号形式的函数，它适合给定的数据样本，可以看作是一个搜索
a mathematical expression (S-expression) from a space of possible S-expressions that can be composed from a set of available functions and arguments.
由一组可用函数和参数组成的可能的 s 表达式空间的数学表达式(s 表达式)。

The appearance of numeric constants (such as the con-stant 1.6527 in the above correct equation) is typical of rela-tions among empirical data from the real world. Thus, we must deal with the problem of discovering coefficients and
数值常数的出现(如上述正确方程中的常数 1.6527)是来自现实世界的经验数据之间的典型关系。因此，我们必须解决发现系数和
constant values while doing symbolic regression. Constants can be created in genetic programming by
常数值可以通过以下方法在遗传编程中创建
adding an ephemeral random constant ~ to the terminal set. During the creation of generation 0, whenever the ephemeral random constant ~ is chosen for an endpoint of the tree, a random number of an appropriate type in a specified range is generated and attached to the tree at that point. For example, in the real-valued symbolic regres-sion problem at hand, the ephemeral random constants are of floating-point type and their range is between -1.000
在终端集合中添加一个短暂的随机常数。在第 0 代的创建过程中，无论何时为树的一个端点选择短暂的随机常数，都会在指定范围内生成一个适当类型的随机数，并在该点附加到树上。例如，在手头的实值符号回归问题中，短暂的随机常数是浮点型的，它们的范围在 -1.000 之间
and +1.000.
和 + 1.000。

This random generation is done anew each time when an ephemeral ~ terminal is encountered, so that the initial random population contains a variety of different random constants of the specified type. Once generated and inserted into the S-expressions of the initial random popu-lation, these constants remain fixed thereafter. However, after the initial random generation, the numerous different random constants will be moved around from tree to tree by the crossover operation. In many instances, these con-stants will be combined via the arithmetic operations in the function set of the problem.

这种随机生成在每次遇到临时终端时都会重新进行，因此初始随机种群包含各种不同的指定类型的随机常数。一旦生成并插入到初始随机种群的 s 表达式中，这些常量在此之后保持不变。然而，在最初的随机生成之后，大量不同的随机常量将通过交叉操作在树与树之间移动。在许多情况下，这些常量会通过问题函数集中的算术运算进行组合。

This 'moving around' and 'combining' of the random
这种随机的“移动”和“组合”
constants is not at all haphazard, but, instead, is driven by the overall goal of achieving ever better levels of fit-ness. For example, a symbolic expression that is a reason-ably good fit to a target function may become a better fit if a particular constant is, for example, decreased slightly. A slight decrease can be achieved in several different ways. For example, there may be a multiplication by 0.90,
常量并不是完全随意的，相反，它是由达到更好的适应性水平的总体目标驱动的。例如，如果某个特定常数稍微减小，那么符号表达式可能变得更适合目标函数。一个轻微的减少可以通过几种不同的方式来实现。例如，可以乘以 0.90,

a division by 1.10, a subtraction of 0.08, or an addition of -0.004. If a decrease of precisely 0.09 in a particular con-stant would produce a perfect fit, a decrease of 0.07 will usually fit better than a decrease of only 0.05. Thus, the relentless pressure of the fitness function in the natural selection process determines both the direction and magni-tude of the adjustments of the original numerical constants. It is thus possible to genetically evolve numeric constants as required to perform a required symbolic regression on numeric data.

除以 1.10，减去 0.08，或加上 -0.004。如果在特定常数中精确地减少 0.09 会产生一个完美的拟合，那么减少 0.07 通常会比仅减少 0.05 更好。因此，自然选择过程中适应度函数的无情压力决定了原始数值常数调整的方向和幅度。因此，根据需要对数值数据进行必要的符号回归，从基因上进化数值常数是可能的。

We first divide the 30-year, 120-quarter period into a 20-year, 80-quarter in-sample period running from 1959:1 to 1978:4 and a 10-year 40-quarter out-of-sample period run-ning from 1979:1 to 1988:4. This allows us to use the first two-thirds of the data to create the model and to then use the last third of the data to test the model.

我们首先将 30 年的 120 个季度划分为 20 年的 80 个季度的抽样期，从 1959:1 到 1978:4，以及 10 年的 40 个季度的抽样期，从 1979:1 到 1988:4。这允许我们使用前三分之二的数据来创建模型，然后使用后三分之一的数据来测试模型。

The first major step in using genetic programming is to identify the set of terminals. The terminal set for this prob-lem is

使用遗传编程的第一个主要步骤是识别终端的集合。这个问题的终端设置是

T = {GNP82, FM2, FYGM3, ~} .
T = { GNP82，FM2，FYGM3，~ }.

The terminals GNP82, FM2, and FGYM 3 correspond to the independent variables of the model and provide access to the values of the time series. In effect, these terminals are functions of the unstated, implicit time variable which ranges over the various quarters.

终端 GNP82，FM2 和 FGYM 3 对应于模型的独立变量，并提供对时间序列值的访问。实际上，这些终端是未说明的，隐含的时间变量的函数，范围在不同的季度。

The second major step in using genetic programming is to identify a set of functions. The set of functions chosen for this problem is

使用遗传编程的第二个主要步骤是识别一组函数。为这个问题选择的函数集是

F = {+, - , *, %, EXP, RLOG }
F = { + ,-，*,% ，EXP，RLOG }

taking 2, 2, 2, 2, 1, and 1 arguments, respectively.

分别采用 2,2,2,2,1 和 1 个参数。

It is necessary to ensure closure by protecting against the possibility of division by zero and the possibility of creating extremely large or small floating-point values. Accordingly, the protected division function % ordinarily returns the quotient; however, if division by zero is attempted, it returns 1.0. The one-argument exponential function EXP ordinarily returns the result of raising e to the power indi-cated by its one argument. If the result of evaluating EXP or any of the four arithmetic functions would be greater than 101~ or less than 10 -l~ then the nominal value 101~ or l0 -1~ respectively, is returned. The protected logarithm function R L O G returns 0 for an argument of 0 and other-wise returns the logarithm of the absolute value of the argument.

有必要通过防止被零除的可能性和创建极大或极小的浮点值的可能性来确保闭包。因此，受保护的除法函数% 通常返回商；然而，如果尝试被零除法，它返回 1.0。一个参数的指数函数 EXP 通常返回将 e 提高到它的一个参数所指示的幂的结果。如果求 EXP 或四个算术函数中的任意一个的结果大于 101 ~ 或小于 10-l ~，则分别返回名义值 101 ~ 或 l0-1 ~ 。受保护的对数函数 r l o g 对于 0 的参数返回 0，否则返回参数绝对值的对数。

Notice that we are not told *a priori* whether the unknown functional relationship between the given observed data (the three independent variables) and the target function (the dependent variable, GD) is linear, polynomial, exponential, logarithmic, non-linear, or otherwise. The unknown functional relationship could be any combi-nation of the functions in the function set. Notice also that we are also not given the known constant value V for the velocity of money. And, notice that we are not told

注意，我们没有先验地被告知给定的观测数据(三个自变量)和目标函数(因变量 GD)之间的未知函数关系是线性的、多项式的、指数的、对数的、非线性的，还是其他的。未知函数关系可以是函数集中函数的任意组合。注意，我们也没有得到货币流通速度的已知常数 v。注意，我们没有被告知

that the 3-month Treasury bill yields (FYGM3) contained in the terminal set and the addition, subtraction, exponen-tial, and logarithm functions are all irrelevant to finding the econometric model for the dependent variable G D of this problem.
终端集合中包含的 3 个月国库券收益率(FYGM3)以及加、减、指数和对数函数都与求解该问题的因变量 g d 的计量经济学模型无关。

The third major step in using genetic programming is identification o f the fitness function for evaluating how good a given computer program is at solving the problem at hand.
使用遗传规划的第三个主要步骤是确定适应度函数，以评估给定的计算机程序解决手头问题的能力。

The fitness o f an S-expression is the sum, taken over the 80 in-sample quarters, of squares of differences between the value of the price level produced by S-expression and the target value o f the price level given by the G D time series. Population size was 500 here.
S 表达式的适合度是取样本季度内 80 个季度内 s 表达式产生的价格水平值与 g d 时间序列给出的价格水平目标值之差的平方的总和。这里的人口规模是 500。

The initial rando m population (generation 0) was, pre-dictably, highly unfit. In one run, the sum of squared errors between the single best S-expression in the popu-lation and the actual G D time series was 1.55. The corre-lation R 2 was 0.49.
最初的随机 m 种群(第 0 代)预计是非常不适合的。在一次运行中，总体中单个最佳 s 表达式与实际 g d 时间序列之间的平方误差总和为 1.55。相关系数 r2 为 0.49。

As before, after the initial rando m population was created, each successive new generation in the population was created by applying the operations of fitness propor - tionate reproduction and genetic recombination (cross-over).
和以前一样，在初始随机 m 群体建立之后，通过适应度比例繁殖和遗传重组(交叉)操作，建立群体中的每一个连续的新世代。

In generation 1, the sum o f the squared errors for the new best single individual in the population improved to 0.50.
在第 1 代中，人群中新的最佳单个个体的平方误差总和提高到 0.50。

In generation 3, the sum o f the squared errors for the new best single individual in the population improved to 0.05. This is approximately a 3t-to-1 improvement over the initial rando m generation. The value of R 2 improved to 0.98. In addition, by generation 3, the best single individual in the population came within 1% of the actual G D time series for 44 of the 80 in-sample points.
在第三代中，群体中新的最佳单个个体的平方误差总和提高到 0.05。比起最初的随机 m 代，这大约是 3 t 比 1 的改进。R 2 的值提高到 0.98。此外，到第 3 代时，在 80 个样本点中的 44 个样本点中，群体中最好的单个个体在实际 g d 时间序列的 1% 以内。

In generation 6, the sum o f the squared errors for the new best single individual in the population improved to 0.027. This is approximately a 2-to-1 improvement over gen-eration 3. The value o f R 2 improved to 0.99.
在第 6 代中，人口中新的最佳单个个体的平方误差总和提高到 0.027。与第三代相比，这大约是 2:1 的改进。R 2 的值提高到 0.99。

In generation 7, the sum o f the squared errors for the new best single individual in the population improved to 0.013. This is approximately a 2-to-1 improvement over gen-eration 6.
在第 7 代，人口中新的最佳单个个体的平方误差总和提高到 0.013。与第 6 代相比，这大约是 2:1 的改进。

In generation 15, the sum o f the squared errors for the new best single individual in the population improved to 0.011. This is an additional improvement over generation
在第 15 代，人口中新的最佳单个个体的平方误差总和提高到 0.011。这是一个额外的进步
7 and represents approximately a 141-to-1 improvement over generation 0. The correlation R 2 was 0.99.
相关系数 r2 为 0.99。

In one run, the best single individual had a sum of squared errors of only 0.009272 over the in-sample period. Figure 14 graphically depicts this best-of-run individual.
在一次运行中，最好的单个个体在样本期间的平方误差总和仅为 0.009272。图 14 图形化地描述了这个最佳运行个体。

This best-of-run individual is equivalent to
这个跑得最好的人相当于

G D - (1.634 * M2)
G d-(1.634 * M2)
GNP82
GNP82

Notice the sub-tree (* - 0 . 40 2 0 - 0 . 583) on the left of this best-of-run individual. This sub-expression evaluates to
注意子树(*-0。4020-0.583)在这个跑得最好的人的左边。这个子表达式计算为

Fig. 14. *Best-of-run individualfor exchange equation problem*
图 14。交换方程问题的最佳运行个体

-+-0.234. The numeric constants - 0 . 4 0 2 0 and - 0 . 58 3 were created in generation 0 by the constant creation process. These two constants are combined into a new constant (+0.234), which, in conjunction with other such con-stants, eventually produces the overall 1.634 constant as the velocity of money.
+-0.234.数值常数 -0。4020 和 -0。583 是通过不断的创造过程在第 0 代创造出来的。这两个常数合并成一个新的常数(+ 0.234)，再加上其他这样的常数，最终得到总的 1.634 常数，即货币流通速度。

Although genetic programming has succeeded in finding an expression that fits the given data rather well, there is always a concern that a fitting technique may be overfitting (i.e. memorizing) the data. If a fitting technique overfits the data, the model produced has no ability to generalize to new combinations of the independent variables and there-fore has little or no predictive or explanatory value. We can validate the model produced from the 80-quarter in-sample period with the data from the 40-quarter out-of-sample period.
尽管遗传编程已经成功地找到了一种表达式，它能很好地适应给定的数据，但是人们总是担心一种拟合技术可能会过度拟合(即记忆)数据。如果拟合技术覆盖了数据，所产生的模型没有能力推广到新的自变量组合，因此很少或没有预测或解释价值。我们可以用 40 个季度的样本期间的数据验证 80 个季度的样本期间产生的模型。

Table 1. *Comparison of in-sample and out-of-sample periods*
表 1. 样本内和样本外时期的比较

Data range 数据范围	1-120 1-120	1-80 1-80	81-120 81-120
R 2 R2	0.993 480 0.993480	0.997 949 0.997949	0.990 614 0.99061 4
Sum of squared error 误差平方和	0.075 388 0.075388	0.009 272 0.009272	0.066 116 0.06611 6

K o z a

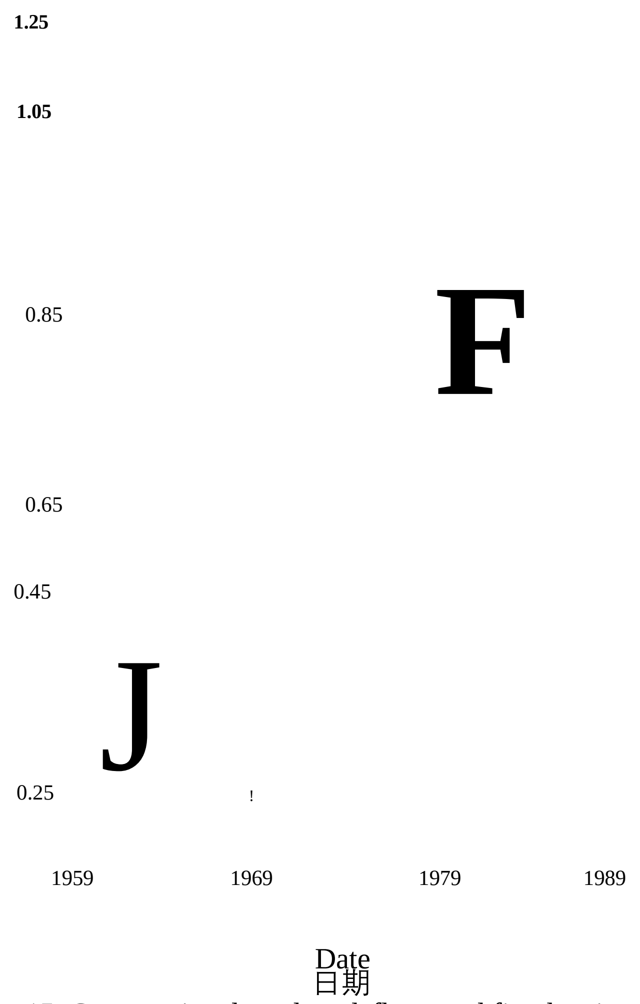

Fig. 15. Gross national product deflator and fitted series computed
图 15。计算的国民生产总值平减指数和拟合系列

from genetically produced model
来自基因产生的模型

Table 1 shows the sum of the squared errors and R 2 for
表 1 显示了平方误差和 r2 的总和

the entire 120-quarter period, the 80-quarter in-sample
整个 120 个季度，样本中的 80 个季度

period, and the 40-quarter out-of-sample period.
期间，以及 40 季度的样本外期间。

Figure 15 shows both the gross national product deflator
图 15 显示了国民生产总值平减指数

GD from 1959:1 to 1988:4 and the fitted GD series cal-
GD 从 1959:1 到 1988:4 和配套的 GD 系列卡路里

culated from the above genetically produced model

and dynamically decomposing the problem into simpler
并动态地将问题分解为更简单的

subproblems.
子问题。

A human programmer writing a computer program to
编写计算机程序的人类程序员

solve a problem often creates a subroutine (procedure,
解决问题常常创建一个子程序(过程,

function) enabling a common calculation to be performed
函数)，使一个共同的计算可以执行

without tediously rewriting the code for that calculation.
而不需要繁琐地重写计算代码。

For example, a programmer who needed to write a pro-
例如，一个程序员需要编写一个 pro

gram for Boolean parity functions of several different
几个不同的布尔奇偶函数

high orders might find it convenient first to write a sub-
高阶可能会发现，首先编写一个子

routine for some lower-order parity function. The code
一些低阶奇偶函数的例程。代码

for this low-order parity function would be called at differ-
因为这个低阶奇偶函数会被调用在不同的-

ent places and with different combinations of arguments
不同的地方和不同的参数组合

from the main program and the results then combined in
从主程序和结果然后结合在一起

the main program to produce the desired higher-order
的主程序，以产生期望的更高阶

parity function. Specifically, a programmer using the
奇偶函数。具体来说，是一个程序员 使用 方法

LISP programming language might first write a function
LISP 编程语言可能会先写一个函数

definition for the odd-2-parity function xor (exclusive-or)
奇 2 奇偶函数 xor (exclusive-or)的定义

as follows:
如下:

```
(defun xor (argO argl)
(defun xor (argO argl)
  (values (or (and argo (not argl))
```

for
根据上述基因生产模型
1959:1 to 1959: 1988:4. The actual GD series is shown as a line
1988:4 实际的 GD 级数显示为一条直线
with dotted points. The fitted GD series calculated from
以点为单位计算的拟合 GD 级数
the above model is an ordinary line.
上面的模型是一条普通的线。

Figure 16 shows the residuals from the fitted GD series
我想也是 16 显示拟合 GD 系列的残差

calculated from the above genetically produced model for
基因产生的模型计算出来的
1959:1 to 1988:4.
1959: 1988:4.

We can further increase confidence that this genetically
我们可以进一步增加信心，这种基因

evolved model is not overfitting the data by dividing the
演化模型不是通过划分数据来过度拟合

same 30-year period into a different set of in-sample and
同样的 30 年时间，变成了一套不同的样本

out-of-sample periods. When we divide the 30-year, 120-
当我们把 30 年，120 年除以
quarter period into a 10-year, 40-quarter out-of-sample
四分之一的时期进入 10 年，40 个季度的样本
period running from 1959:1 to 1968:4 and a 20-year, 80-
从... 开始 1959:1 到 1968:4,20 年，80 年
quarter in-sample period running from 1969:1 to 1988:4,
四分之一在样本期间运行 1969:1 至 1969 年 1988:4,

we obtain a virtually identical model. See Koza (1992a).
我们获得了几乎相同的模型。参见 Koza (1992a)。

6. Hierarchical automatic function definition - ll - parity function
分级自动功能

(值(或(和 argo (不是 argl)))

(and (not argO) argl)))).
(和(不是阿尔戈) argl))。

This function definition (called a 'defun' in LISP) does
这个函数定义(在 LISP 中称为“ defun”)是这样做的
four things. First, it assigns a name, xor, to the function
首先，它给函数分配一个名称 xor
being defined thereby permitting subsequent reference to
定义，从而允许随后引用
it. Second, it identifies the argument list of the function
其次，它确定了函数的参数列表
being defined, namely the list (argO argl) containing two
定义，即包含两个
dummy variables (formal parameters) called argO and
虚拟变量(形式参数)称为 argO 和
argl. Third, it contains a body which performs the work
第三，它包含一个执行这项工作的机构
of the function. Fourth, it identifies the value to be
函数。第四，它确定了应该是什么值
returned by the function. In this example, the single value
在这个例子中，单个值
to be returned is emphasized via an explicit invocation of
通过显式地调用
the 'values' function. This particular function definition
“值”函数。这个特殊的函数定义
has two dummy arguments, returns only a single value,
有两个虚拟参数，只返回一个值,
has no side effects, and refers only to the two local
没有副作用，仅指两个局部
dummy variables (i.e. it does not refer to any of the
虚拟变量(即它不引用任何
actual variables of the overall problem contained in the

定义 -ll-parity 函数

A key goal in machine learning and artificial intelligence is
机器学习和人工智能的一个关键目标是
to facilitate the solution of a problem by automatically
自动解决问题

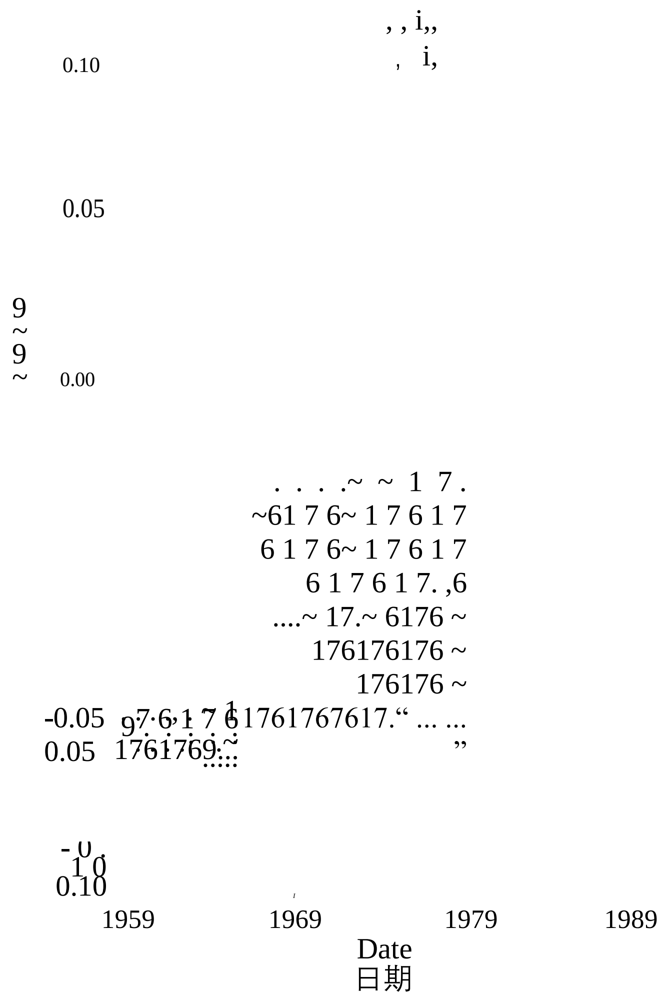

Fig. 16. *Residuals between the gross national product deflator and fitted series computed from genetically produced model*
图 16。国民生产总值平减指数与根据基因生产模型计算的拟合序列之间的残差

中所包含的整个问题的实际变量
'main' program). However, in general, defined functions
然而，一般来说，定义的函数
may have any number of arguments (including no argu-
可能有任何数量的争论(包括没有争论)
ments), may return multiple values (or no values), may or
可以返回多个值(或者没有值)，可以或者
may not perform side effects, and may or may not expli-
可能不会产生副作用，也可能不会解释
citly refer to the actual (global) variables of the main
城市参考的实际(全局)变量的主要
program.
程序。

Once the function xor is defined, it may then be
一旦定义了 xor 函数，它就可能是
repeatedly called with different instantiations of its
对象的不同实例化反复调用
arguments from more than one place in the main pro-
来自不止一个地方的论点主要支持
gram. For example, a programmer who needed the
例如，一个程序员需要

even-4-parity at some point in the main program might
甚至 -4 奇偶校验在主程序的某个点上也可能
write
: 写

(xor (xor dO dl) (not (xor d2 d3))).
(xor (xor dO dl)(不是(xor d2 d3)))。

Function definitions exploit the underlying regularities and symmetries of a problem by obviating the need to rewrite lines of essentially similar code. A function
函数定义通过避免重写基本上相似的代码行来利用问题的潜在规则性和对称性。一个函数

definition is especially efficient when it is repeatedly called with different instantiations of its arguments. However, the importance of function definition goes well beyond efficiency. The process of defining and calling a function, in effect, decomposes the problem into a hierarchy of sub-problems.
定义在用其参数的不同实例重复调用时特别有效。然而，函数定义的重要性远远超出了效率。实际上，定义和调用函数的过程将问题分解为一系列子问题。

The ability to extract a reusable subroutine is potentially very useful in many domains. Consider the problem of dis-covery of a neural network to recognize patterns presented as an array of pixels. Suppose the solution of a pattern recognition problem requires discovery of a particular fea-ture (e.g. a line end) within the 3 x 3 pixel region in the upper left corner of an 8 x 8 array o f pixels and also requires discovery of that same feature within a 3 x 3 pixel region in the lower left corner of the overall array. Existing neural net paradigms can successfully discover the useful feature among the nine pixels Pll, P12, P13, P21, P22, P23, P31, P32, P33 in the upper left corner of an 8 x 8 array of pixels and can independently rediscover the same useful feature among the nine pixels P6~, P62, P63, P16, P71, P72, P73, P81, P82, P83 in the lower left corner of the overall array. But existing neural net paradigms do not provide a way to discover the commo n *feature just once*, to generalize the feature so that it is not rigidly expressed in terms o f par-ticular pixels but is parametrized by its position, and then to reuse the generalized feature detector to recognize occur-rences of the feature in different 3 x 3 pixel regions within the array. That is, existing paradigms do not provide a way to discover a function of nine dummy variables *just once* and to call that function twice (once with P11, ... , P33 as arguments and once with P61,... ,P83 as arguments). Such an ability would amoun t to discovering a nine-input subassembly of neurons with appropriate weights, making a copy of the entire subassembly, implanting the copy else-where in the overall neural net, and then connecting nine different pixels as inputs to the subassembly in its new location in the overall neural net.

提取可重用子程序的能力在许多领域都是非常有用的。考虑发现神经网络的问题，以识别呈现为像素数组的模式。假设一个模式识别问题的解决方案需要在一个 8 x 8 像素阵列的左上角的 3 x 3 像素区域内发现一个特定的特征(例如线端)，并且还需要在整个阵列的左下角的 3 x 3 像素区域内发现相同的特征。现有的神经网络范式能够成功地发现 8 × 8 像素阵列左上角 9 个像素中的 Pll、 P12、 P13、 P21、 P22、P23、 P31、 P32、 P33 之间的有用特征，并能够独立地重新发现整个阵列左下角 9 个像素中的相同有用特征。但是，现有的神经网络模型并没有提供一次性发现常见的 n 个特征的方法，也没有提供一次性发现常见的 n 个特征的方法，也没有提供一次性发现常见的 n 个特征的方法，也没有提供一次性发现常见的 n 个特征的方法，也没有提供一次性发现常见的 n 个特征的方法，也没有提供一次性发现常见的 n 个特征的方法。也就是说，现有的范式没有提供一种方法来发现一个由 9 个虚拟变量组成的函数，只有一次，并且两次调用该函数(一次用 P11，...，P33 作为参数，一次用 P61，...，P83 作为参数)。这种能力相当于发现一个具有适当权重的九个输入神经元的子组件，复制整个子组件，将复制品植入整个神经网络中的其他位置，然后将九个不同的像素作为输入连接到子组件在整个神经网络中的新位置。

Hierarchical automatic function definition can be imple-mented within the context of genetic programming by establishing a constrained syntactic structure for the indi-vidual S-expressions in the population (Koza, 1992a). Each individual S-expression in the population contains one (or more) function-defining branches and one (or more) 'main' result-producing branches. The result-producing branch may call the defined functions. One defined function ma y hierarchically refer to another already-defined function (and potentially even itself), although such hierarchical or recursive references will not be used in this article.

通过建立群体中个体 s 表达式的约束句法结构，可以在遗传规划的背景下实现递阶自动函数定义(Koza，1992a)。人群中的每个个体 s- 表达式包含一个(或多个)功能定义分支和一个(或多个)“主要”结果生成分支。结果生成分支可以调用已定义的函数。一个已定义的函数在层次上引用另一个已定义的函数(甚至可能引用自己)，尽管本文不会使用这种层次或递归引用。

6.1. *Learning* ***the even-parity function without hierarchical automatic function definition***

6.1 学习偶校验函数没有层次化的自动函数定义

In order to establish the facilitating benefits of hierarchical automatic function definition in genetic programming, we
为了证明递阶自动函数定义在遗传规划中的便利性，我们提出了一种基于递阶自动函数定义的遗传规划方法

first solve some benchmark problems without using hier-archical automatic function definition.

首先解决一些基准问题，而不使用分层自动函数定义。

The Boolean even-parity function of k Boolean argu-ments returns T (true) if an even number of its arguments are T, and otherwise returns N I L (false).

如果 k Boolean 参数的偶数为 t，则布尔偶校验函数返回 t (true)，否则返回 n i l (false)。

In applying genetic programming to the even-parity function of k arguments, the terminal set T consists of the k Boolean arguments DO, D 1, D2, .. . involved in the prob - lem, so that

将遗传规划应用于 k 参数的偶奇偶函数时，终端集 t 由 k 布尔参数 DO，d1，D2，。..包含在问题中，所以

T = {DO, D1, D2, ...}.

T = { DO，D1，D2，... }.

The function set F for all the examples herein consists of the following computationally complete set of four two-argument primitive Boolean functions:

这里所有例子的函数集 f 由以下四个双参数基元布尔函数的计算完整集合组成:

F = {AND, OR, N A N D , NOR} .

F = { AND，OR，n a n d，NOR }.

The Boolean even-parity functions appear to be the most difficult Boolean functions to find via a blind rando m gen-erative search of S-expressions using the above function set

布尔偶校验函数似乎是最难找到的布尔函数通过盲随机 m 生成搜索的 s 表达式使用上述函数集

F and the terminal set T. F o r example, even though there are only 256 different Boolean functions with three argu-ments and one output, the Boolean even-3-parity function is so difficult to find via a blind rando m generative search that we did not encounter it at all after randomly generat-ing 10 000 000 S-expressions using this function set F and terminal set T. In addition, the even-parity function appears to be the most difficult to learn using genetic pro-gramming using the function set F and terminal set T above (Koza, 1992a).

例如，尽管只有 256 个不同的布尔函数有三个参数和一个输出，但是通过盲随机 m 生成搜索很难找到布尔偶 -3 奇偶函数，所以我们使用这个函数集 f 和终端集 t 随机生成了 1000000000 个 s 表达式后，根本没有遇到它。此外，使用上述函数集 f 和终端集 t 进行遗传程序设计时，偶宇称函数似乎是最难学习的(Koza，1992a)。

In applying genetic programming to the problem o f learning the Boolean even-parity function of k arguments, the 2 k combinations of the k Boolean arguments constitute an exhaustive set of fitness cases for learning this function. The standardized fitness o f an S-expression is the sum, over these 2 k fitness cases, of the Hamming distance (error) between the value returned by the S-expression and the correct value of the Boolean function. Standardized fitness ranges between 0 and 2~; a value closer to zero is better. The raw fitness is equal to the number of fitness cases for which the S-expression is correct (i.e. 2 k minus standardized fit-ness); a higher value is better.

将遗传规划应用于 k 参数的布尔偶校验函数的学习问题中，k 布尔参数的 2 k 组合构成了学习该函数的一组穷举的适应性案例。S 表达式的标准化适应度是在这 2 k 适应度情况下，s 表达式返回的值与布尔函数的正确值之间的汉明距离(误差)的和。标准化适应度范围介于 0 和 2 之间; 接近于 0 的值更好。原始适应度等于 s 表达式正确的适应情况的个数(即 2k 减去标准适应度)，值越大越好。

We first consider how genetic programming would solve the problems of learning the even-3-parity function (three-argument Boolean rule 105), the even-4-parity function (four-argument Boolean rule 38505), and the even-5-parity function (five-argument Boolean rule 1 771476585). In identifying these k-argument Boolean functions in this way, we are employing a numbering scheme wherein the value of the function for the 2 k combi-nations of its k Boolean arguments are concatenated into a 2k-bit binary number and then converted to the equivalent decimal number. F o r example, the 23 : 8 values of the even-3-parity function are 0, 1, 1, 0, 1, 0, 0, and 1 (going from the fitness case consisting o f three true arguments to the fitness case consisting of three false arguments). Since

我们首先考虑遗传规划如何解决学习偶 -3 奇偶函数(三参数布尔规则 105)、偶 -4 奇偶函数(四参数布尔规则 38505)和偶 -5 奇偶函数(五参数布尔规则 1771476585)的问题。在以这种方式识别这些 k 参数布尔函数时，我们使用了一种编号方案，其中将函数的 k 布尔参数的 2 k 组合的值连接成一个 2 k 位二进制数，然后转换成等效的十进制数。例如，偶 -3 奇偶校验函数的 23:8 值是 0,1,1,0,1,0,0 和 1(从包含三个真参数的适应情况到包含三个假参数的适应情况)。因为

104

011010012 = 10510, the even-3-parity function is referred to as three-argument Boolean rule 105.
011010012 = 10510，偶 -3 奇偶校验函数称为三参数布尔规则 105。

The terminal set T for the even-3-parity problem consists
偶数 -3 奇偶问题的终端集 t 包括

of
的资料

T = {DO, D1, D2}.
T = { DO，D1，D2}.

In one run of genetic programming using a population size of 4000 (the value of M used consistently in this sec-tion, except as otherwise noted), genetic programming dis-covered the following S-expression containing 45 points (i.e. 22 functions and 23 terminals) with a perfect value of raw fitness of 8 (out of a possible value of 23 = 8) in gen-eration 5:

在一次使用 4000 人口规模的遗传规划中(本节中一致使用的 m 值，除非另有说明)，遗传规划发现以下 s 表达式包含 45 个点(即 22 个函数和 23 个终端)，在第 5 代中原始适应度的完美值为 8(23 = 8) :

(AND (OR (OR DO (NOR 9 2 D1)) 0 2) (AND (NAND
(AND (OR (OR DO (NOR 92 D1))02)(AND (NAND

(NOR (NOR DO 92) (AND (AND D1 D1) D1))
(NOR (NOR DO 92)(AND (AND D1 D1) D1))

(NAN D (OR (AND D0 D1) 92) DO)) (OR (NAND
(NAN d (或(和 D0 D1)92) DO))(或(NAND

(AND DO D2) (OR (NOR DO (OR 0 2 D0)) D1))
(AND DO D2)(OR (NOR DO (OR 02 D0)) D1))

(NAND (NAND D1 (NAND DO D1)) D2)))).
(NAND (NAND D1(NAND DO D1)) D2)).

We then considered the even-4-parity function. In one run, genetic programming discovered a program contain-ing 149 points with a perfect value of raw fitness of 16 (out of 24 = 16) in generation 24.

然后我们考虑了偶 -4 奇偶函数。在一次运行中，遗传编程发现了一个包含 149 个点的程序，在第 24 代中，原始适应度的完美值为 16(24 = 16)。

Figure 17 presents two curves, called the performance curves, relating to the even-3-parity function over a series of runs. The curves are based on 66 runs with a population size M of 4000 and a maximum number of generations to be run G of 51. The rising curve in Fig. 17 shows, by gen-eration, the experimentally observed cumulative prob-ability of success, *P (M , i)* , of solving the problem by generation i (i.e. finding at least one S-expression in the population which produces the correct value for all 23=- 8 fitness cases). As can be seen, the experimentally observed value of the cumulative probability of success, *P(M , i),* is 91% by generation 9 and 100% by generation 21 over the 66 runs. The second curve in Fig. 17 shows, by generation, the number of individuals that must be pro-cessed, *I (M , i, z),* to yield, with probability z, a solution to the problem by generation *i. I (M , i, z)* is derived from the experimentally observed values of *P (M , i).* Specifically, *I (M , i, z)* is the product of the population size M, the gen-eration number i, and the number of independent runs *R(z)* necessary to yield a solution to the problem with prob-ability z by generation i. In turn, the number of runs *R(z)* is given by

图 17 显示了两条曲线，称为性能曲线，与一系列运行中的偶数 -3 奇偶函数有关。这些曲线基于 66 次运行，人口大小 m 为 4000，最大代数 g 为 51。图 17 中的上升曲线显示，经过一代，实验观察到的成功的累积概率，p (m，i)，通过一代解决问题(即在总体中找到至少一个 s 表达式，该表达式对所有 23 =-8 个适应性情况产生正确的值)。可以看到，实验观察到的累积成功概率 p (m，i)在第 9 代为 91%，在第 21 代为 100%。图 17 中的第二条曲线显示，按代数，必须处理的个体数目，i (m，i，z)，以产生概率 z，按代数 i (m，i，z)的问题的解决方案是由 p (m，i)的实验观察值得出的。具体地说，i (m，i，z)是群体大小 m，世代数 i，以及产生一个问题的解所必需的独立运行次数 r (z)的乘积。反过来，r (z)的运行次数由

V log(l--z) 1
V l o g (l- z) 1
R(z) = [log (1 - *P (M , i))* ʹ
R (z) = [log (1-p (m，i))ʹ

where the square brackets indicates the ceiling function for rounding up to the next highest integer. The probability z will be 99% herein.
其中方括号表示四舍五入到下一个最高整数的上限函数。在这里，z 的概率为 99%。

As can be seen, the *I (M , i, z)* curve reaches a minimum value at generation 9 (highlighted by the light dotted
可以看到，i (m，i，z)曲线在第 9 代达到了一个最小值

Koz
a
Koz
a
-
1,200,00
0 O
一百二
十万
!

lOO,
卢,

-*- P(M,i)
-*- P (m, i)

t
~
T
~

I(M, i, z)
I (m, i, z)
,600,000
60 万

50

so,ooo
)
所
以,
哦)

f 9 1
F91

,0
0

0 25 50
G e n e r a t i o n
快点, 快点

Fig . 17. *Performance curvesfor even-3-parity function showing that it is sufficient to process 80 000 individuals to yield a solution with 99% probability with genetic programming*
无花果。17.偶数 -3 奇偶函数的性能曲线表明，它足以处理 80000 个个体，产生一个解决方案的 99% 的概率与遗传编程

vertical line). For a value of *P (M , i)* of 91%, the number of independent runs *R(z)* necessary to yield a solution to the problem with a 99% probability by generation i is 2. The two summary numbers (i.e. 9 and 80 000) in the oval indicate that if this problem is run through to generation 9 (the initial random generation being counted as gen-eration 0), processing a total of 80000 individuals (i.e. 4000 x 10 generations x 2 runs) is sufficient to yield a solution to this problem with 99% probability. This number 80 000 is a measure of the computational effort necessary to yield a solution to this problem with 99% probability.

垂直线)。对于 91% 的 p (m, i)值，产生一个 99% 概率的问题的解所需的独立运行次数 r (z)为 2。椭圆形中的两个汇总数字(即 9 和 80000)表明，如果这个问题被运行到第 9 代(最初的随机世代被计算为第 0 代)，处理总共 80000 个个体(即 4000x10 个世代 x2 运行)足以以 99% 的概率产生这个问题的解。这个数字 80000 是一个必要的计算努力的度量，以产生一个 99% 概率的问题的解决方案。

Figure 18 shows similar performance curves for the even-4-parity function based on 60 runs. The experimentally observed cumulative probability of success, *P (M , i),* is 35% by generation 28 and 45% by generation 50. The *I (M , i,z)* curve reaches a minimum value at generation

图 18 显示了基于 60 次运行的偶 4 奇偶校验函数的类似性能曲线。实验观察到的累积成功概率 p (m, i)在第 28 代为 35%， 在第 50 代为 45% 。I (m, i, z)曲线在产生时达到最小值

28. For a value of *P (M , i)* of 35%, the number of runs *R(z)* is 11. The two numbers in the oval indicate that if this problem is run through to generation 28, processing a total of 1 276000 (i.e. 4000 x 29 generations x 11 runs) individuals is sufficient to yield a solution to this problem with 99% probability. Thus, according to this measure of

对于 35% 的 p (m, i)， 运行次数 r (z)为 11。椭圆形中的两个数字表明，如果这个问题一直运行到第 28 代，总共处理 1276000 个个体(即 4000x29 代 x11 次运行)就足以产生这个问题的解，概率为 99% 。因此，根据这个测量

5o **20,000,000**
5,2000 万

~ 2 5 10.000.0110
~ 2 万 5 千 1 万 0110

~ **m P(M,0 I(M, i,z)**
M p (m, 0 i (m, i, z)
o

o
25 50
2550
G e n e r a t i o n
快点，快点

Fig . 18. *Performance curves for even-4-parity function showing that it is sufficient to process 1276 000 individuals to yield a solution with 99% probability with genetic programming*
无花果。18.对偶数 -4 字称函数的性能曲线表明，它足以处理 1276000 个个体，产生一个解决方案的 99% 的概率与遗传规划

computational effort, the even-4-parity problem is about 16 times harder to solve than the even-3-parity problem.

在计算上的努力下，偶数 -4 宇称问题比偶数 -3 宇称问题难解 16 倍。

We are unable to extend directly this comparison of the computational effort necessary to solve the even-parity problem with increasing numbers of arguments with our chosen population size of 4000. When the even-5-parity function was run with a population size of 4000 and each run arbitrarily stopped at our chosen maximum number G = 51 of generations to be run, no solution was found after 20 runs. (Solutions might well have been found if we had continued the run, but we did not do this.) Even after increasing the population size of 8000 (with G = 51), we did not get a solution until our eighth run. This solution contained 347 points.

我们无法直接扩展这种计算工作量的比较，这种计算工作量是解决偶等价问题所必需的，而我们选择的总体规模是 4000。当偶数 -5 奇偶函数在人口大小为 4000 时运行，并且每次运行任意停止在我们选择的最大数目 g = 51 的要运行的世代时，20 次运行后没有发现任何解。(如果我们继续运行，很可能已经找到了解决方案，但是我们没有这样做甚至在增加了 8000 个人口(g = 51)之后，我们直到第八次运行才得到一个解决方案。这个解决方案包含了 347 个点。

Notice that the structural complexity (i.e. the total number of function points and terminal points in the S-expression) of the solutions produced in these three cited runs dramatically increased with an increasing number of arguments (i.e. structural complexity was 45, 149, and 347, respectively, above for the 3-, 4-, and 5-parity functions).

请注意，在这三次引用运算中产生的解的结构复杂度(即 s 表达式中的函数点和终端点的总数)随着参数数目的增加而显著增加(即结构复杂度分别为 45,149 和 347，上述 3-，4-和 5 奇偶函数)。

The population size of 4000 is undoubtedly not optimal for any particular parity problem and is certainly not opti-mal for all sizes of parity problems. Nonetheless, it is clear that learning the even-parity functions with increasing numbers of arguments requires dramatically increasing computational effort and that the structural complexity of the solutions become increasingly large.

4000 人的人口规模无疑不是任何特定平等问题的最佳人口规模，也肯定不是所有规模的平等问题的最佳人口规模。尽管如此，很明显，学习偶奇偶函数的参数数量越来越多，需要大大增加计算量，解的结构复杂性也越来越大。

6.2. *Hierarchical automatic function definition*

2. 分层自动函数定义

The inevitable increase in computational effort and struc-tural complexity for solving parity problems of order greater than 4 could be controlled if we could discover the underlying regularities and symmetries of this problem and then hierarchically decompose the problem into more tractable sub-problems. Specifically, we need to discover a function parametrized by dummy variables that would be helpful in decomposing and solving the problem.

如果我们能够发现问题的潜在规律和对称性，然后将问题分层分解为更易处理的子问题，就可以控制求解大于 4 阶奇偶问题的计算量和结构复杂度的不可避免的增加。具体来说，我们需要发现一个由虚拟变量参数化的函数，这将有助于分解和解决问题。

A human programmer writing code for the even-3-parity or even-4-parity functions would probably choose to call upon either the odd-2-parity function (also known as the exclusive-or function XOR) or the even-2-parity function (also known as the equivalence function EQV). For the even-5-parity function and parity functions with addi-tional arguments, our programmer would probably also want to call upon either the even-3-parity (3-argument Boolean rule 105) or the odd-3-parity (3-argument Boolean rule 150). These lower-order parity functions

为偶 -3 奇偶或偶 -4 奇偶函数编写代码的人类程序员可能会选择调用奇 2 奇偶函数(也称为异或函数)或偶 2 奇偶函数(也称为等价函数 EQV)。对于偶数 -5 奇偶校验函数和带有附加参数的奇偶校验函数，我们的程序员可能还想调用偶数 -3 奇偶校验(3 参数布尔规则 105)或奇数 -3 奇偶校验(3 参数布尔规则 150)。这些低阶奇偶函数

would greatly facilitate writing code for the higher-order parity functions. None of these low-order parity functions is, of course, in our original set F of available primitive Boolean functions.

将极大地方便为高阶奇偶函数编写代码。当然，这些低阶奇偶函数中没有一个在我们原始的可用基元布尔函数集 f 中。

The potentially helpful role of dynamically evolving useful 'building blocks' in genetic programming has been

在遗传编程中动态演化有用的“构建块”的潜在有用的作用已经被

recognized for some time (Koza, 1990). However, when we talk about 'hierarchical automatic function definition' in this article, we are not contemplating merely defining a function in terms of a sub-expression composed of particu-lar fixed terminals (i.e. actual variables) of the problem. Instead, we are contemplating defining functions *para-metrized* by dummy variables (formal parameters). Speci-fically, if the exclusive-or function XOR were being automatically defined during a run, it would be a version of XOR parametrized by two dummy variables (perhaps called ARGO and ARG1), not a mere call to XOR with par-ticular fixed actual variables of the problem (e.g. DO and D1). When this parametrized version of the XOR function is called, its two dummy variables ARGO and ARG1 would be instantiated with two specific values, which would either be the values of two terminals (i.e. actual variables of the problem) or the values of two expressions (each composed ultimately of terminals). For example, the exclusive-or function XOR might be called via (XOR DO D1) on one occasion and via (XOR D2 D3) on another occasion. On yet another occasion, XOR might be called via

(Koza，1990).然而，当我们在本文中讨论“层次化自动函数定义”时，我们并不仅仅考虑根据由问题的特定固定终端(即实际变量)组成的子表达式来定义函数。相反，我们正在考虑定义由虚拟变量(形式参数)构成的函数。具体来说，如果在运行过程中自动定义了排他或函数 XOR，那么它将是由两个虚拟变量(可能称为 ARGO 和 ARG1)参数化的 XOR 版本，而不仅仅是带有问题的特定固定实际变量(例如 DO 和 D1)的 XOR 调用。当调用这个 XOR 函数的参数化版本时，它的两个虚拟变量 ARGO 和 ARG1 将使用两个特定值进行实例化，这两个值可以是两个终端(即问题的实际变量)的值，也可以是两个表达式(每个表达式最终由终端组成)的值。例如，排他或函数 XOR 可以在一个场合调用 via (XOR DO D1)，在另一个场合调用 via (XOR D2 D3)。在另一种情况下，XOR 可以通过

(XOR (AND D1 D2) (OR DO D2)),
(XOR (AND D1 D2)(OR DO D2))

where the two arguments to XOR are the values returned by the expressions (AND D1 D2) and (OR DO D2), respec-tively. Each of these expressions is ultimately composed of the actual variables (i.e. terminals) of the problem.

其中 XOR 的两个参数分别是表达式(AND D1 D2)和(OR DO D2)返回的值。每个表达式最终都由问题的实际变量(即终端)组成。

Moreover, when we talk about 'automatic' and 'dynamic' function definition, the goal is to evolve dynamically a dual structure containing both function-defining branches and result-producing (i.e. value-returning) branches by means of natural selection and genetic operations. We expect that genetic programming will dynamically evolve poten-tially useful function definitions during the run and also dynamically evolve an appropriate result-producing 'main' program that calls these automatically defined functions.

此外，当我们谈论“自动”和“动态”功能定义时，我们的目标是通过自然选择和遗传操作，动态地演化出一个包含功能定义分支和结果产生分支(即价值返回分支)的双重结构。我们期望遗传编程能够在运行过程中动态地演化出潜在有用的函数定义，并且动态地演化出一个调用这些自动定义函数的产生结果的“主”程序。

Note that many existing paradigms for machine learning and artificial intelligence do define functional subunits automatically and dynamically during runs (the specific terminology, of course, being specific to the particular para-digm). For example, when a set of weights is discovered enabling a particular neuron in a neural network to per-form some subtask, that learning process can be viewed as a process of defining a function (i.e. a function taking the values of the specific inputs to that neuron as argu-ments and returning an output signal, perhaps a 0 or 1). Note, however, that the function thus defined can be called only once from only one particular place within the neural network. It is called only in the specific part of the neural net (i.e. the neuron) where it was created and it is called only with the original, fixed set of inputs to that specific neuron. Note also that existing paradigms for neural networks do not provide a way to re-use the set of weights discovered in that part of the network in other

请注意，许多现有的机器学习和人工智能范例确实在运行期间自动和动态地定义了功能子单元(当然，特定的术语是特定于特定范例的)。例如，当发现一组权重，使神经网络中的某个特定神经元能够执行某个子任务时，该学习过程可以被视为定义一个函数的过程(即将特定输入的值作为参数传递给该神经元并返回输出信号，也许是 0 或 1 的函数)。然而，请注意，这样定义的函数只能从神经网络中的一个特定位置调用一次。它只在神经网络的特定部分(即神经元)被调用，在那里它被创建，它只被调用与原始的，固定的集合输入到特定的神经元。还要注意的是，现有的神经网络模型并没有提供一种方法来重新使用在该部分网络中发现的权重集合

parts of the network where a similar subtask must be performed on a different set of inputs. The recent work of Gruau (1992) on recursive solutions to Boolean functions is a notable exception.
网络的一部分，其中一个类似的子任务必须在不同的输入集上执行。Gruau (1992)最近关于布尔函数的递归解决方案的工作是一个值得注意的例外。

Hierarchical automatic function definition can be implemented within the context of genetic programming by establishing a constrained syntactic structure (Koza, 1992a, Chapter 19) for the individual S-expressions in the population in which each individual contains one or more function-defining branches and one or more 'main' result-producing branches which may call the defined functions.

在遗传程序设计的背景下，可以通过建立一个约束的句法结构(Koza，1992a，第 19 章)来实现分层的自动功能定义，该句法结构用于每个个体包含一个或多个功能定义分支和一个或多个‘主要’结果生成分支的群体中的个体 s 表达式，这些分支可以调用已定义的功能。

The number of result-producing branches is determined by the nature of the problem. Since Boolean parity func-tions return only a single Boolean value, there would be only one 'main' result-producing branch to the S-expres-sion in the constrained syntactic structure required.

结果生成分支的数量取决于问题的性质。由于布尔奇偶函数只返回一个布尔值，因此在所需的约束语法结构中，s 表达式只有一个“主”结果生成分支。

We usually do not know *a priori* the optimal number of functions that will be useful for a given problem or the optimal number of arguments for each such function; however, considerations of computer resources (time, virtual memory usage, CONSing, garbage collection, and memory fragmentation) necessitate that choices be made. Additional computer resources are required for each addi-tional function definition. There is a considerable increase in the computer resources required to support the ever-larger S-expressions associated with each larger number of arguments. There will usually be no advantage to having defined functions that take more arguments than there are terminals in the problem. When Boolean func-tions are involved, there is no advantage to evolving one-argument function definitions (since the only four one-argument Boolean functions are either in our function set already or constant-valued functions).

我们通常先验地不知道对给定问题有用的函数的最佳数量或每个此类函数的参数的最佳数量；然而，考虑到计算机资源(时间、虚拟内存使用、 CONSing、垃圾收集和内存碎片)，必须做出选择。每个额外的函数定义都需要额外的计算机资源。计算机资源需要大量增加，以支持与更大数量的参数相关联的越来越大的 s 表达式。定义了比问题中的终端更多的参数的函数通常是没有好处的。当涉及到布尔函数时，演化一个参数的函数定义没有任何好处(因为仅有的四个一个参数的布尔函数要么在我们的函数集中，要么在常数值函数中)。

Thus, for the Boolean even-4-parity problem, it would seem reasonable to permit one two-argument function definition and one three-argument function definition within each S-expression. Thus, each individual S-expression in the population would have three branches. The first (left-

因此，对于 Boolean 偶数 -4 奇偶校验问题，在每个 s 表达式中允许一个两个参数的函数定义和一个三个参数的函数定义似乎是合理的。因此，人口中的每个单独的 s 表达式将有三个分支。第一个(左)

Koza

most) branch permits a two-argument function definition (defining a function called ADF0); the second (middle) branch permits a three-argument function definition (defining a function called ADF1); and the third (right-most) branch is the result-producing branch. The first two branches are function-defining branches which may or may not be called upon by the result-producing branch.

Most)分支允许一个双参数函数定义(定义一个称为 ADF0 的函数)；第二个(中间)分支允许一个三参数函数定义(定义一个称为 ADF1 的函数)；第三个(最右边)分支是结果生成分支。前两个分支是定义函数的分支，可能被结果生成分支调用，也可能不被调用。

Figure 19 shows an abstraction of the overall structure of an S-expression with two function-defining branches and one result-producing branch. There are 11 'types' of points in each individual S-expression in the population for this problem. The first eight types are an invariant part of each individual S-expression.

图 19 显示了一个 s 表达式的整体结构的抽象，它有两个函数定义分支和一个结果生成分支。在这个问题的群体中，每个单独的 s 表达式有 11 个“类型”的点。前八个类型是每个单独的 s 表达式的不变部分。

The 11 types are as follows:

11 种类型如下:

1. the root (which will always be the place-holding PROGN function);

   根(永远是保持位置的 PROGN 函数)；
2. the top point DEFUN of the function-defining branch for ADF0;

   ADF0 函数定义分支的顶点 DEFUN;
3. the name ADF0 of the function defined by this first function-defining branch;

   由第一个函数定义分支定义的函数的名称 ADF0;
4. the argument list (ARGO ARGI) of ADF0;

   ADF0 的参数列表(ARGO ARGI)；
5. the top point DEFUN of the function-defining branch for ADF1;

   ADF1 函数定义分支的顶点 DEFUN;
6. the name ADF1 of the function defined by this second function-defining branch;

   由第二个函数定义分支定义的函数的名称 ADF1;
7. the argument list (ARGO ARG1 ARG2) of ADF1;

   ADF1 的参数列表(ARGO ARG1 ARG2)；
8. the top point VALUES of the result-producing branch for the individual S-expression as a whole;

   结果生成分支的顶点值作为一个整体用于单个 s- 表达式;
9. the body of ADF0;

   ADF0 的主体;
10. the body of ADF1;

    ADF1 的主体;
11. the body of the 'main' result-producing branch.

    “主要”结果产生分支的主体。

Syntactic rules of construction govern points of types 9, 10, and 11.

构造的句法规则控制类型 9、10 和 11 的点。

For points of type 9, the body of ADF0 is a composition of functions from the given function set F and terminals from the terminal set A2 of two dummy variables, namely A2 = {ARGO, ARGI}.

对于类型 9 的点，ADF0 的主体是来自给定函数集 f 的函数和来自两个虚变量即 A2 = { ARGO，ARGI }的终端集 A2 的终端的函数的组合。

Fig. 19. *Abstraction of the overall structure of an S-expression with two function-defining branches and the one result-producing branch*

图 19。具有两个函数定义分支和一个结果生成分支的 s 表达式的总体结构的抽象

For the points of type 10, the body of ADF1 is a composition of functions from the original given function set F *along with* ADF0 and terminals from the set A3 of three dummy variables, namely A3 = {ARGO, ARG1, ARG2}. Thus, the body of ADF1 is capable of calling upon ADF0.

对于类型 10 的点，ADF1 的主体是来自原始给定函数集 f 的函数以及 ADF0 和来自三个虚变量集 A3(即 A3 = { ARGO，ARG1，ARG2})的终端的组合。因此，ADF1 的主体能够调用 ADF0。

For the points of type 11, the body of the result-producing branch is a composition of terminals (i.e. actual variables of the problem) from the terminal set T, namely T = {DO, D1, D2, D3}, as well as functions from the set F3. F3 contains the four original functions from the func-tion set F as well as the two-argument function ADF0 defined by the first branch and the three-argument func-tion ADF 1 defined by the second branch. That is, the func-tion set F3 is

对于类型 11 的点，结果生成分支的主体是来自终端集 t (即 t = { DO，D1，D2，D3})的终端(即问题的实际变量)以及来自集合 F3 的函数的组合。F3 包含函数集 f 中的四个原始函数，以及由第一个分支定义的双参数函数 ADF0 和由第二个分支定义的三参数函数 adf1。也就是说，函数集 F3 是

F3 = {AND, OR, NAND, NOR, ADF0, ADF1},
F3 = { AND，OR，NAND，NOR，ADF0，ADF1} ,

taking two, two, two, two, two, and three arguments, respectively. Thus, the result-producing branch is capable of calling the two defined functions ADF0 and ADF1.
分别取两个，两个，两个，两个和三个参数。因此，结果生成分支能够调用两个定义的函数 ADF0 和 ADF1。

When the overall S-expression in Fig. 19 is evaluated, the PROGN evaluates each branch; however, the value(s) returned by the PROGN consists only of the value(s) returned by the VALUES function in the final result-producing branch.

当计算图 19 中的整个 s 表达式时，PROGN 计算每个分支; 然而，PROGN 返回的值仅由最终结果生成分支中 VALUES 函数返回的值组成。

Note that one might consider including the terminals from the terminal set T (i.e. the actual variables of the prob-lem) in the function-defining branches; however, we do not do so here.

请注意，我们可以考虑在函数定义分支中包含来自终端集 t (即问题的实际变量)的终端; 但是，我们在这里不这样做。

In what follows, genetic programming will be allowed to evolve two function definitions in the function-defining branches of each S-expression and then, at its discretion, to call one, two, or none of these defined functions in the result-producing branch. We do not specify what functions will be defined in the two function-defining branches. We do not specify whether the defined functions will actually be used. As we have already seen it is possible to solve this problem without any function definition by evolving the correct program in the result-producing branch. We do not favour one function-defining branch over the other. We do not require that a function-defining branch use all of its available dummy variables. The structure of all three branches is determined by the combined effect, over many generations, by the selective pressure exerted by the fitness measure and by the effects of the operations of Darwinian fitness proportionate reproduction and crossover.

接下来，遗传编程将被允许在每个 s 表达式的函数定义分支中发展两个函数定义，然后根据自己的判断，在结果生成分支中调用一个、两个或者不调用这些定义的函数。我们没有指定在两个函数定义分支中定义什么函数。我们没有指定定义的函数是否会被实际使用。正如我们已经看到的，在没有任何函数定义的情况下，通过在结果生成分支中进化正确的程序来解决这个问题是可能的。我们不喜欢一个函数定义分支胜过另一个。我们不要求函数定义分支使用所有可用的虚拟变量。所有三个分支的结构是由多代以来的综合效应、适应性测度施加的选择压力以及达尔文适应性比例繁殖和交叉操作的影响所决定的。

Since a constrained syntactic structure is involved, we must create the initial random generation so that every individual S-expression in the population has the syntactic structure specified by the syntactic rules of construction presented above. Specifically, every individual S-expres-sion must have the invariant structure represented by the eight points of types 1 through 8. Specifically, the bodies of ADF0 (type 9), ADF1 (type 10), and the result-producing branch (type 11) must be composed of the func-

由于涉及到一个有约束的句法结构，我们必须创建一个初始的随机生成，这样群体中的每个 s- 表达式都具有上面给出的构造句法规则所指定的句法结构。具体来说，每个单独的 s- 表达式必须具有由类型 1 到 8 的 8 个点表示的不变结构。具体地说，ADF0(类型 9)、 ADF1(类型 10)和结果生成分支(类型 11)的主体必须由 func-组成

tions and terminals specified by the above syntactic rules of construction.
上述结构句法规则所规定的句子和终端。

Moreover, since a constrained syntactic structure is involved, we must perform structure-preserving crossover so as to ensure the syntactic validity of all offspring as the run proceeds from generation to generation. Structure-preserving crossover is implemented by first allowing the selection of the crossover point in the first parent to be any point from the body of ADF0 (type 9), ADF1 (type 10), or the result-producing branch (type 11). However, once the crossover point in the first parent has been selected, the crossover point of the second parent must be of the same type (i.e. types 9, 10, or 11). This restriction on the selection of the crossover point of the second parent assures syntactic validity of the offspring.

此外，由于涉及一个约束的句法结构，我们必须执行结构保持交叉，以确保所有子代的句法有效性，因为运行从一代到另一代。保持结构的交叉是通过首先允许第一个父系中的交叉点选择为来自 ADF0(类型 9)、ADF1(类型 10)或结果生成分支(类型 11)主体的任何点来实现的。然而，一旦选择了第一个父类的交叉点，第二个父类的交叉点必须是相同的类型(即类型 9、10 或 11)。这种对第二亲本交叉点选择的限制确保了后代的句法有效性。

6.3. *Even-4-parityfunction*

3. 偶 -4 平价函数

Each S-expression in the population for solving the even-4-parity function has one result-producing branch and two function-defining branches, each permitting the definition of one function of three dummy variables.
求解偶 4 奇偶函数的群体中的每个 s 表达式都有一个结果生成分支和两个函数定义分支，每个分支允许定义三个虚变量的一个函数。

In one run of the even-4-parity function, the following 100%-correct solution containing 45 points (not counting the invariant points of types 1 through 8) with a perfect value of 16 for raw fitness appeared on generation 4:
在偶数 -4 奇偶函数的一次运行中，第 4 代出现了以下 100% 正确的解，其中包含 45 个点(不包括类型 1 至类型 8 的不变点)，原始适应度的完美值为 16:

```
(PROGN (DEFUN ADF0 (ARGO ARG1  ARG2)
 (PROGN (DEFUN ADF0(ARGO ARG1 ARG2)

              (NOR (NOR ARG2 ARGO)
              (NOR (NOR ARG2 ARGO)

               (AND ARGO ARG2)))
                 (和 ARGO ARG2))

 (DEFUN ADF1  (ARGO ARG1 ARG2)
 (DEFUN ADF1(ARG1 ARG2)

    (NAND  (ADF0 ARG2 ARG2 ARGO)
    (NAND (ADF0 ARG2 ARG2 ARGO)

       (NAND  (ADF0 ARG2 ARG1  ARG2)
       (NAND (ADF0 ARG2 ARG1 ARG2)

          (ADF0 (OR ARG2 ARG1)
           (ADF0(或 ARG2 ARG1)

            (NOR ARGO ARG1)
            (NOR ARGO ARG1)

            (ADF0 ARG1 ARGO
            (ADF0 ARG1 ARGO

              ARG2)))))
              ARG2)))

 (VALUES
 (价值观

    (ADF0 (ADF1  D1 D3 DO)
    (ADF0(adf1d1d3do)

       (NOR (OR D2 D3) (AND D3 D3))
       (NOR (或 d2d3)(和 d3d3))

       (ADF0 D3 D3 D2)))).
       (ADF0 D3 D3 D2)).
```

The first branch of this best-of-run S-expression is a function definition establishing the defined function ADF0 as the two-argument exclusive-or (XOR) function. The definition of ADF0 ignores one of the available dummy variables, namely ARG1. The second branch of the S-expression calls upon the defined function ADF0 (i.e. XOR) to define ADF1. This second branch **appears**
这个最佳运行 S-expression 的第一个分支是一个函数定义，它将定义的函数 ADF0 建立为两个参数排他或(XOR)函数。ADF0 的定义忽略了一个可用的虚拟变量，即 ARG1。S-表达式的第二个分支调用已定义的函数 ADF0(即异或)来定义 ADF1。第二个分支出现了

Fig. 20. *Hierarchy (lattice) o f function definitions*
图 20。函数定义的层次(格子)

to use all three available dummy variables; however, it reduces to the two-argument equivalence function EQV. The result-producing (i.e. third) branch of this S-expression uses all four terminals and both ADF0 and ADF1 to solve the even-4-parity problem. This branch reduces to
使用所有三个可用的虚拟变量，然而，它减少到两个参数的等价函数 EQV。这个 s- 表达式的结果生成(即第三个)分支使用所有四个终端和 ADF0 和 ADF1 来解决偶 -4 奇偶问题。这个分支可以简化为

(ADF0 (ADF1 Dt DO) (ADF0 D3 D2)).
(ADF0(adf1dt DO)(adf0d3d2)).

which is equivalent to
相当于

(XOR (EQV D1 DO) (XOR D3 D2)).
(XOR (EQV D1 DO)(XOR D3 D2)).

That is, genetic programming decomposed the even-4-parity problem into two different parity problems of lower order (i.e. XOR and EQV).
也就是说，遗传规划将偶 4 奇偶问题分解为两个不同的低阶奇偶问题(即异或和 EQV)。

Figure 20 shows the hierarchy (lattice) of function definitions used in this solution to the even-4-parity problem. Note also that the second of the two functions in this decomposition (i.e. EQV) was defined in terms of the first (i.e. XOR).

图 20 显示了偶数 -4 奇偶问题解决方案中使用的函数定义的层次结构(格子)。还要注意，这个分解中的两个函数中的第二个(即 EQV)是根据第一个(即异或)定义的。

Note that we did not specify that the exclusive-or XOR function would be defined in ADF0, as opposed to, say, the equivalence function, the if-then function, or any other Boolean function. Similarly, we did not specify what would be evolved in ADF1. Genetic programming created the two-argument defined functions ADF0 and ADF1 on its own to help solve this problem. Having done this, genetic programming then used ADF0 and ADF1 in an appropriate way in the result-producing branch to solve the problem. Notice that the 45 points above are considerably fewer than the 149 points con-tained in the S-expression cited earlier for the even-4-parity problem.

请注意，我们没有指定排他或异或函数将在 ADF0 中定义，而不是等价函数、 if-then 函数或任何其他布尔函数。同样，我们没有指定 ADF1 中将演变什么。遗传编程自己创建了两个参数定义的函数 ADF0 和 ADF1 来帮助解决这个问题。这样做之后，遗传编程在结果生成分支中以适当的方式使用 ADF0 和 ADF1 来解决这个问题。请注意，上面的 45 个点比之前引用的 s 表达式中包含的 149 个点要少得多。

Figure 21 presents the performance curves based on 23 runs for the even-4-parity with hierarchical automatic function definition. The cumulative probability of success *P (M , i)* is 91% by generation 10 and 100% by generation
图 21 显示了基于 23 次运行的性能曲线，这些运行是为了使用分层自动函数定义的偶 -4 奇偶校验。成功的累积概率 p (m，i)在第 10 代为 91%，在第 100 代为 100%
50. The two numbers in the oval indicate that if this prob-lem is run through to generation 10, processing a total of 88000 individuals (i.e. 4000 • 11 generations • 2 runs) is sufficient to yield a solution to this problem with *9 9 %* probability.
椭圆形中的两个数字表明，如果这个问题一直进行到第 10 代，处理总共 88000 个个体(即 4000•11 代•2 次)就足以产生这个问题的解，概率为 9.9% 。

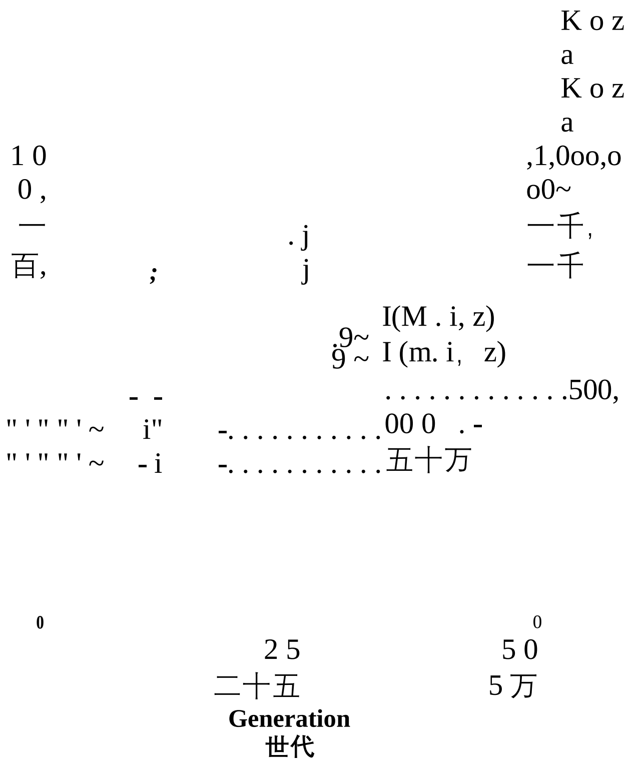

Fig. 21. *Performance curves for the even-4-parity problem show that it is sufficient to process 88 000 individuals to yield a solution with hierarchical automatic function definition*

图 21。偶数 -4 奇偶问题的性能曲线表明，它足以处理 88000 个个体，以产生一个解决方案与分层自动功能定义

6.4. *Even 5-parity function*
甚至 5 奇偶函数

Each program in the population for solving the even-5-parity function (and all higher-order parity functions herein) has one result-producing branch and two function-defining branches, each permitting the definition of one function of four dummy variables.

求解偶数 -5 奇偶函数(以及本文中所有高阶奇偶函数)的群体中的每个程序都有一个结果生成分支和两个函数定义分支，每个分支允许定义一个由四个虚变量组成的函数。

In one run of the even-5-parity problem, the 100%-correct solution contains 160 points and emerged on generation 12. The first branch is equivalent to the four-argument Boolean rule 50 115 which is an even-2-parity function that ignores two of the four available dummy variables. The second branch is equivalent to the four-argument Boolean rule 38 250, which is equivalent to

在偶数 -5 奇偶问题的一次运行中，100% 正确的解包含 160 个点，并出现在第 12 代。第一个分支相当于四个参数的 Boolean 规则 50115，它是一个偶数 -2 奇偶校验函数，它忽略了四个可用的虚变量中的两个。第二个分支相当于四个参数的布尔规则 38250，相当于

```
(OR (AND (NOT ARG2) (XOR ARG3 ARGO))
(OR (AND (NOT ARG2)(XOR ARG3 ARGO))
```

```
(AND ARG2    (XOR ARG3 (XOR ARG1
(和 ARG2(XOR ARG3(XOR ARG1

                    ARGO)))).
                   阿尔戈)))。
```

Notice that this rule is not a parity function of any kind. The result-producing (i.e. third) branch calls on defined functions ADF0 and ADF1 and solves the problem.

注意，这个规则不是任何类型的奇偶函数。结果生成(即第三个)分支调用已定义的函数 ADF0 和 ADF1 并解决问题。

The even 5-parity problem can be similarly solved with 99% probability with genetic programming using hier-archical automatic function definition by processing a total of 144 000 individuals.

通过对 144000 个个体进行分层自动函数定义，利用遗传规划的 99% 的概率，可以类似地解决偶数 5 奇偶问题。

6.5. *Parity functions with 7 to 10 arguments*
奇偶函数有 7 到 10 个参数

The even 6-, and 7-parity problems can be similarly solved with 99% probability with genetic programming using hier-archical automatic function definition by processing a total of 864 000, and 1 440 000 individuals, respectively.

采用层次自动函数定义的遗传规划方法，分别处理 864000 个个体和 1440000 个个体，可以同样以 99% 的概率解决偶数 6 奇偶问题和 7 奇偶问题。

The 8-, 9-, and 10-parity problems can be similarly solved using hierarchical automatic function definition. Each problem was solved within the first four runs. We did not perform sufficient additional runs to compute a perfor-mance curve for these higher-order parity problems.

8-，9-和 10 奇偶问题可以类似地解决使用分层自动函数定义。每个问题都在前四次运行中得到解决。我们没有执行足够的额外运行来计算这些高阶奇偶问题的性能曲线。

6.6. *Even-ll-parity function*
6. 偶数奇偶函数

In one run of the even-l 1-parity function, the following best-of-generation individual containing 220 points and attaining a perfect value of raw fitness of 2048 appeared in generation 21:
在一次偶 11 奇偶函数的运行中，第 21 代出现了下列最佳代数个体，其中包含 220 个点，并获得了 2048 年原始适应度的完美值:

```
(PROGN (DEFUN ADF0 (ARGO ARG1 ARG2 ARG3)
(PROGN (DEFUN ADF0(ARGO ARG1 ARG2 ARG3)
        (NAND (NOR (NAND (OR ARG2 ARG1)
        (NAND (NOR (NAND (OR ARG2 ARG1))
        (NAND ARG1 ARG2)) (NOR (OR ARG1
        (NAND ARG1 ARG2))(NOR (或 ARG1
        ARGO) (NAND ARG3 ARG1))) (NAND
        ARGO)(NAND ARG3 ARG1))(NAND
        (NAND (NAND (NAND ARG1 ARG2)
        (NAND (NAND (NAND ARG1 ARG2)
        ARG1) (OR ARG3 ARG2)) (NOR (NAND
        ARG1)(或 ARG3 ARG2))(NOR (NAND
        ARG2 ARG3) (OR ARG1 ARG3)))))
        ARG2 ARG3)(或 ARG1 ARG3)))
    (DEFUN ADF1 (ARGO ARG1 ARG2 ARG3)
    (DEFUN ADF1(ARG1 ARG2 ARG3)
        (ADF0 (NAND (OR ARG3 (OR ARGO
        ARGO)) (AND (NOR ARG1 ARG1)
        (ADF0(NAND (或 ARG3(或 ARGO
        ARGO))(AND (NOR ARG1 ARG1)
        (ADF0 ARG1 ARG1 ARG3 ARG3)))
        (NAND (NAND (ADF0 ARG2 ARG1
        (ADF0 ARG1 ARG3 ARG3))(NAND
        (NAND (ADF0 ARG2 ARG1
        ARGO ARG3) (ADF0 ARG2 ARG3 ARG3
        ARGO ARG3)(ADF0 ARG2 ARG3 ARG3
        ARG2)) (ADF0 (NAND ARG3 ARGO)
        ARG2)(ADF0(NAND ARG3 ARGO)
        (NOR ARGO ARG1) (AND ARG3 ARG3)
        (NOR ARGO ARG1)(和 ARG3 ARG3)
        (NAND ARG3 ARGO))) (ADF0 (NAND
        (NAND ARG3 ARGO))(ADF0(NAND
        (OR ARGO ARGO) (ADF0 ARG3 ARG1
        ARG2 ARGO)) (ADF0 (NOR ARGO
        ARGO) (NAND ARGO ARG3) (OR ARG3
        ARG2) (ADF0 ARG1 ARG3 ARGO
        (ADF0(NOR ARGO ARGO)(NAND ARGO
        ARG3)(OR ARG3 ARG2)(ADF0 ARG1
        ARG3 ARGO)
        ARGO)) (NOR (ADF0 ARG2 ARG1 ARG2
        ARGO))(NOR (ADF0 ARG2 ARG1 ARG2
        ARGO) (NAND ARG3 ARG3)) (AND
        ARGO)(NAND ARG3 ARG3))(和
          (AND ARG2 ARG1) (NOR ARG1 ARG2)))
            (和 ARG2 ARG1)(NOR ARG1 ARG2))
        (AND (NAND (OR ARG3 ARG2) (NAND
        (和(NAND (或 ARG3 ARG2)(NAND
        ARG3 ARG3)) (OR (NAND ARG3 ARG3)
        ARG3 ARG3)(或(NAND ARG3 ARG3)
        (AND ARGO ARGO)))))
        (和 ARGO ARGO)))
    (VALUES
    (价值观
        (OR (ADF1 D1 DO (ADF0 (ADF1 (OR
        (或(ADF1 D1 DO (ADF0(ADF1(OR
        (NAND D1 D7) D1) (ADF0 D1 D6 D2 D6)
        (ADF1 D6 D6 D4 D7) (NAND D6 D4))
        (ADF1 (ADF0 D9 D3 D2 D6) (OR D10 D1)
        (ADF1 D3 D4 D6 D7) (ADF0 D10 D8 D9
        D5)) (ADF0 (NOR D6 D9) (NAND D1
        (ADF1(ADF0 D9 D3 D2 D6)(OR D10
        D1)(ADF1 D3 D4 D7)(ADF0 D10 D8 D9
        D5)(ADF0(NOR D6 D9)(NAND D1)
```

D10) (ADF0 D10 D5 D3 D5) (NOR D8
D10)(ADF0 D10 D5 D3 D5)(NOR D8

D2)) (OR D6 (NOR D1 DG))) D1) (NOR
D2))(或 D6(NOR D1 DG))) D1)(NOR

(NAND Dl D10) (ADF0 (OR (ADF0 D6
(NAND Dl D10)(ADF0(或(ADF0 D6

D2 D8 D4) (OR D4 D7)) (NOR D10 D6)
D2 D8 D4)(或 D4 D7))(非 D10 D6)

(NOR D1 D2) (ADF1 D3 D7 D7 DG)))))).
(NOR D1 D2)(ADF1 D3 D7 D7 DG))).

The first branch of this S-expression defined the four-argument defined function ADF0 (four-argument Boolean rule 50 115) which ignored two of its four arguments. ADF0 is equivalent to the even-2-parity function, namely
这个 s 表达式的第一个分支定义了四个参数定义的函数 ADF0(四个参数布尔规则 50115)，它忽略了其四个参数中的两个。ADF0 等价于偶数 -2 奇偶函数，即

(EQV ARG1 ARG2).
(EQV ARG1 ARG2).

The second branch defined a four-argument defined function ADF1 which is equivalent to the even-4-parity function. Substituting the definitions of the defined functions ADF0 and ADF1, the result-producing (i.e. third) branch becomes:
第二个分支定义了一个四参数定义函数 ADF1，它等价于偶数 -4 奇偶函数。取代定义函数 ADF0 和 ADF1 的定义，结果生成(即第三)分支变成:

(OR (EVEN-4-PARITY
(OR (EVEN-4-PARITY

D1
D1

DO
DO 做

(EVEN-2-PARITY
(偶数 -2-平价

(EVEN-4-PARITY
(EVEN-4-PARITY

(EVEN-2-PARITY D3 D2)
(偶数 -2-奇偶校验 d3d2)

(OR D10 D1)
(或 D10 D1)

(EVEN-4-PARITY D3 D4 D6 D7)
(甚至 -4-奇偶校验 D3 D4 D6 D7)

(EVEN-2-PARITY D8 D9))
(EVEN-2-PARITY D8 D9)

(EVEN-2-PARITY (NAND D1 D10)
(偶数 -2 奇偶校验(nandd1d10)

(EVEN-2-PARITY D5 D3)))
(EVEN-2-PARITY D5 D3))

DU
DU

(NOR (NAND D1 D10)
(NOR (NAND D1 D10)

(EVEN-2-PARITY (NOR D10 D6)
(甚至 -2 奇偶校验(NOR D10 D6)

(NOR D1 D2))))
(NOR D1 D2))

which is equivalent to the target even-11-parity function. Note that the even-2-parity function (ADF0) appears six times in this solution and that the even-4-parity function (ADF1) appears three times. Note that this entire solution for the even-11-parity function contains only 220 points (compared with 347 points for the solution to the *mere even-5-parity* without hierarchical automatic function definition).
等价于目标偶数 -11 奇偶函数。请注意，偶 -2 奇偶校验函数(ADF0)在此解中出现 6 次，偶 -4 奇偶校验函数(ADF1)出现 3 次。请注意，偶 -11 奇偶校验函数的整个解决方案仅包含 220 个点(相比之下，没有层次自动函数定义的偶 -5 奇偶校验的解决方案为 347 个点)。

Figure 22 shows the simplified version of the result-producing branch of this best-of-run individual for the even-ll-parity problem. As can be seen, the even-ll-parity problem was decomposed into a composition of even-2-parity functions and even-4-parity functions.

图 22 显示了这个最佳运行个体的结果生成分支的简化版本。可以看出，偶极子问题被分解为偶极子函数和偶极子函数的组合。

Fig. 22. *The best-of-run individualfrom generation 21 of one run of the even-l l-parity problem is a composition of even-2-parity and even-4-parity functions*
图 22。一次偶 -1l 奇偶问题的第 21 代的最佳运行个体是由偶 -2- 奇偶和偶 -4- 奇偶函数组成的

We found the above solution to the even- 11-parity problem on our first completed run. The search space of 11-argument Boolean functions returning one value is of size 22~ ~ 10616. The even-11-parity problem was solved by decomposing into parity functions of lower orders.

在我们第一次完成的运行中，我们找到了解决偶 11 奇偶问题的上述方法。返回一个值的 11 个参数布尔函数的搜索空间大小为 22 ~ 10616。偶数 -11 奇偶问题通过分解成低阶奇偶函数来解决。

6.7. *Summary o f hierarchical automatic function definition*
6.7 层次自动函数定义的总结

Thus, the problem of learning various higher order even-parity functions can be solved with the technique of hier-archical automatic function definition in the context of genetic programming. Moreover, as can be seen in Table 2, the technique of hierarchical automatic function defi-nition facilitates the solution of these problems. That is, when problems are decomposed into a hierarchy of func-tion definitions and calls, many fewer individuals must be processed in order to yield a solution to the problem. More-over, the solutions discovered are comparatively smaller in terms of their structural complexity.
因此，在遗传规划的背景下，利用高阶自动函数定义技术可以解决学习各种高阶偶校验函数的问题。此外，如表 2 所示，分层自动函数定义技术有助于解决这些问题。也就是说，当问题被分解成函数定义和调用的层次结构时，必须处理更少的个体才能产生问题的解决方案。更重要的是，所发现的解决方案在结构复杂性方面相对较小。

Automatic function definition has also been applied to the problem of discovery of impulse response functions (Koza *et al.*, 1993).
自动函数定义也被应用于脉冲响应函数的发现问题 (Koza 等，1993)。

7. Additional examples of genetic programming
7. 遗传规划的其他例子

Genetic programming can be applied in many additional
遗传编程可以应用于许多其他领域

Table 2. *Number of individuals I(M, i, z) required to be processed to yield a solution to various even-parity problems with 99% probability - with and without hierarchical automatic function definition*
表 2。需要处理的个体数 i (m，i，z)的数量，以产生各种具有 99% 概率奇偶性的问题的解-有或没有分层自动函数定义

Size of 大小 parity function 奇偶函数	Without hierarchical 没有层次 automatic function 自动功能 definition 定义	With hierarchical 等级森严 automatic function 自动功能 definition 定义
3	80000	
4	1276000	88000
5		144000
		864 000
6		864000
		1 440 000
7		144 万

problem domains, including the following:
问题领域，包括:

9 evolution of a subsumption architecture for control-ling a robot to follow walls or move boxes (Koza, 1992d; Koza and Rice, 1992b);
9 包容式架构的演变，用于控制机器人跟随墙壁或移动箱子(Koza，1992d; Koza 和 Rice，1992b) ;
9 discovering inverse kinematic equations to control the movement of a robot arm to a designated target point; 9 emergent behaviour (e.g. discovering a computer pro-gram which, when executed by all the ants in an ant colony, enables the ants to locate food, pick it up, carry it to the nest, and drop pheromones along the way so as to recruit other ants into cooperative
9 发现逆运动学方程，控制机器人手臂向指定目标点的移动; 9 紧急行为(例如发现一个计算机程序，当蚁群中的所有蚂蚁执行时，使蚂蚁能够定位食物，捡起食物，带到巢穴，并沿途投放信息素，以招募其他蚂蚁合作
behaviour);
行为) ;
9 symbolic integration, symbolic differentiation, and symbolic solution of general functional equations (including differential equations with initial conditions); 9 planning (e.g. navigating an artificial ant along a trail, developing a robotic action sequence that can stack an arbitrary initial configuration of blocks into a specified
9 一般函数方程的符号积分、符号微分和符号解(包括带初始条件的微分方程) ; 9 规划(例如沿着小路引导一只人工蚂蚁，开发一个机器人动作序列，可以将任意初始配置的块堆叠到指定的位置)
order);
命令) ;
9 generation of highlentropy sequences of random numbers;
9 代随机数的高熵序列;

9 induction of decision trees for classification;
9 归纳决策树进行分类;
9 optimization problems (e.g. finding an optimal food foraging strategy for a lizard);
9 个优化问题(例如为蜥蜴找到最佳的觅食策略) ;
9 sequence induction (e.g. inducing a recursive compu-tational procedure for generating sequences such as the Fibonacci sequence);
9 序列归纳(例如，归纳一个递归计算过程来生成序列，如 Fibonacci 序列) ;
9 automatic programming of cellular automata;
9 元胞自动机的自动编程;
9 finding minimax strategies for games (e.g. differential pursuer-evader games, discrete games in extensive form) by both evolution and co-evolution;
9 通过进化和协同进化找出博弈的极大极小策略(如微分追逃博弈、广义离散博弈) ;
9 automatic programming (e.g. discovering a compu-tational procedure for solving pairs of linear equations, solving quadratic equations for complex roots, and discovering trigonometric identities);
9 自动编程(例如发现求解线性方程组对的计算程序、求解复根的二次方程组和发现三角恒等式) ;
9 simultaneous architectural design and training of neural networks (Koza and Rice, 1991).
9 同时进行神经网络的建筑设计和训练(Koza 和 Rice，1991)。

Additional information and examples can be found in Koza (1992a).
其他信息和例子可以在 Koza (1992a)中找到。

8. Conclusions
8. 遗传算法和遗传算法组合研讨会的结论

We have shown that m a n y seemingly different problems in machine learning and artificial intelligence can be viewed as requiring the discovery of a compute r p r o g r a m that pro - duces some desired output for particular inputs. We have also shown that the recently developed genetic program - ming paradig m described herein provides a way to search for a highly fit individual compute r program . The tech-nique o f hierarchical automatic function definition can facilitate the solution o f problems .
我们已经证明，在机器学习和人工智能中，有许多看似不同的问题，可以被看作是需要发现一个计算机 r p r o g r a m，从而为特定的输入产生一些期望的输出。我们还表明，最近开发的遗传程序-明范例 m 在这里描述提供了一种方法来搜索一个高度适合的个人计算 r 程序。分层自动功能定义的技术可以促进问题的解决。

Acknowledgements
鸣谢

Christopher Jones prepared the figures for the econometric problem . James P. Rice o f the Knowledge Systems L a b o r a - tory at Stanford University created all the other figures in this article and did all or some o f the compute r program - ming on a Texas Instruments Explorer computer for various problems in this article.
克里斯托弗琼斯为计量经济学问题准备了数据。斯坦福大学知识系统研究所的 James p. Rice 创造了本文中的所有其他数据，并在 Texas Instruments Explorer 计算机上为本文中的各种问题编写了全部或部分计算程序。

References
参考文献

Belew, R. and Booker, L. (EDS) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA.
(EDS)(1991).第四届国际遗传算法会议记录。摩根 ·考夫曼，圣马特奥，加利福尼亚州。

Citibank (1989). *CITIBASE: Citibank Economic Database (Machine Readable Magnetic Data File), 1946-Present.* Citi-
花旗银行(1989)。花旗银行经济数据库(机读磁性数据文件)，1946 年至今
bank N.A., New York.
Bank n.a。，纽约。

Cramer, N. L. (1985). A representation for the adaptive gen-eration of simple sequential programs. In *Proceedings of an of the Workshop on Combinations of Genetic Algorithms and Neural Networks 1992,* ed. J. D. Schaffer and D. Whitley.
神经网络 1992 年，编辑 J.D.Schaffer 和 D.Whitley。
International Conference on Genetic Algorithms and Their Applications, ed. J. Grefenstette. Lawrence Erlbaum, Hills-dale, NJ.
北卡罗来纳州克莱默(1985)。简单顺序程序的自适应生成的表示。在遗传算法及其应用国际会议的会议记录中,。格兰芬斯特。Lawrence Erlbaum，Hills-dale，NJ 劳伦斯 ·埃尔鲍姆，新泽西州。

Davidor, Y. (1991). *Genetic Algorithms and Robotics.* World Scientific, Singapore.
戴维多，y。(1991)，《遗传算法与机器人学》，新加坡。

Davis, L. (EO) (1987). *Genetic Algorithms and Simulated Annealing.*
(1987)。遗传算法与模拟退火。
Pitman, London.
Pitman，London Pitman，伦敦。

Davis, L. (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York.
戴维斯，l。(1991)。遗传算法手册。范诺斯兰德莱茵霍尔德，纽约。

Forrest, S. (zo). (1990). *Emergent Computation." Self-Organizing, Collective, and Cooperative Computing Networks.* MIT
Forrest，s. (zo)。(1990)《紧急计算》，《自组织，集体和合作计算网络》，麻省理工学院
Press, Cambridge, MA.
马萨诸塞州剑桥出版社。

Fujiki, C. and Dickinson, J. (1987). Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma. In
Fujiki，c。and Dickinson，j。(1987)。使用遗传算法生成 LISP 源代码来解决囚徒困境。译注:
Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, ed. J. Grefenstette. Lawrence Erlbaum, Hillsdale, NJ.
遗传算法及其应用: 第二届国际遗传算法会议记录。Grefenstette.Lawrence Erlbaum，新泽西州希尔斯代尔。

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization,*
(1989)，《搜索、优化中的遗传算法》，
and Machine Learning, Addison-Wesley, Reading, MA. Goldberg, D. E., Korb,. B. and Deb, K. (1989). Messy genetic
机器学习，Addison-Wesley，Reading，MA。Goldberg，D.e.，Korb,.B 和 Deb，k。(1989)。凌乱的遗传
algorithms: motivation, analysis, and first results. *Complex Systems,* 3, 493-530.
算法: 动机、分析和初步结果。复杂系统，3,493-530。

Gruau, F. (1992). Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In *Proceedings*
(1992)。具有细胞重写发育过程的布尔神经网络的遗传综合

The IEEE Computer Society Press.
Ieee 计算机协会出版社。

Hallman, J. J., Porter, R. D. and Small, D. H. (1989). *M2per Unit of Potential GNP as an Anchor for the Price Level.* Board of Governors of the Federal Reserve System. Staff Study 157, Washington, DC.

霍尔曼，J.j.，波特，R.d. 和小 D.h. (1989)。M2per Unit of Potential GNP as a Anchor for the Price Level 每单位潜在国民生产总值作为价格水平的锚。美国联邦储备系统董事会。职员研究 157，华盛顿特区。

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

《自然与人工系统的适应》，密西根大学出版社，密西根州安阿伯。

Holland, J. H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning." An Artificial Intelligence Approach, Volume II,* ed. R. S. Michalski *et al.* pp. 593-623. Morgan Kaufmann, Los Altos, CA.

荷兰，J.h. (1986)。逃避脆弱性: 通用学习算法应用于并行规则系统的可能性。在机器学习中。”A Artificial intelligence Approach，Volume II，ed 人工智能方法，第二卷。R.s. Michalski 等，第 593-623 页。摩根考夫曼，洛斯阿尔图斯，加利福尼亚州。

Holland, J. H., Holyoak, K. J., Nisbett, R. E. and Thagard, P. A. (1986). *Induction: Processes of Inference, Learning, and Dis-covery.* MIT Press, Cambridge, MA.

Holland，J.h.，Holyoak，K.j.，Nisbett，R.e. and Thagard，P.a. (1986).归纳: 推理，学习和分析的过程。麻省理工学院出版社，剑桥，马萨诸塞州。

Koza, J. R. (1990). *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Stanford University Computer Science Depart-ment technical report STAN-CS-90-1314.

柯扎(1990)。遗传程序设计: 基因繁殖计算机程序解决问题的范例。斯坦福大学计算机科学系技术报告 STAN-CS-90-1314。

Koza, J. R. (1992a). *Genetic Programming." On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA.

Koza，J.r。(1992a)，《遗传编程》，《关于通过自然选择进行计算机编程》，麻省理工学院出版社，剑桥，马萨诸塞州。

Koza, J. R. (1992b). Genetic programming: genetically breeding populations of computer programs to solve problems. In *Dynamic, Genetic, and Chaotic Programming,* ed. B. Soucek and the IRIS Group. John Wiley, New York.

柯扎(1992b)。遗传编程: 用计算机程序进行遗传育种来解决问题。在动态，遗传和混沌编程，编辑。Soucek 和 IRIS 集团。John Wiley，纽约。

Koza, J. R. (1992c). Hierarchical automatic function definition in genetic programming. In *Proceedings of Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Vail, Colorado 1992,* ed. D. Whitley. Morgan Kaufmann, San Mateo, CA.

Koza，J.r。(1992c)。遗传编程中的分层自动函数定义。在《遗传算法和分类器系统基础研讨会进程》中，韦尔，科罗拉多，1992，编辑。惠特利。Morgan Kaufmann，San Mateo，CA 摩根考夫曼，圣马特奥，加州。

Koza, J. R. (1992d). Evolution of subsumption using genetic programming. In *Proceedings of European Conference on Artificial Life, Paris, December 1991,* ed. P. Bourgine and F. Varela. MIT Press, Cambridge, MA.

柯扎(1992 年 d)。使用遗传编程的包容进化。《欧洲人工生命会议进程》，巴黎，1991 年 12 月。布尔根和 f。瓦雷拉。麻省理工学院出版社，马萨诸塞州剑桥。

Koza, J. R. and Keane, M. A. (1990a). Cart centering and broom balancing by genetically breeding populations of control strategy programs. In *Proceedings of International Joint Conference on Neural Networks, Washington, January 15-19, 1990.* Volume I, pp. 198 201. Lawrence Erlbaum, Hillsdale, NJ.

Koza，J.r. and Keane，M.a. (1990a).通过控制策略程序的遗传育种种群的购物车中心和扫帚平衡。1990 年 1 月 15-19 日，华盛顿，神经网络国际联席会议。第一卷，198201 页。劳伦斯 ·埃尔鲍姆，新泽西州希尔斯代尔。

Koza, R. and Keane, M. A. (1990b). Genetic breeding of non-
(1990b)。非生物的遗传育种
linear optimal control strategies for broom balancing. In *Pro-ceedings of the Ninth International Conference on Analysis and Optimization of Systems. Antibes, France, June, 1990,* pp. 47 - 56. Springer-Verlag, Berlin.
扫帚平衡的线性最优控制策略。在第九届系统分析和优化国际会议的议事录中。昂蒂布，法国，1990 年 6 月，第 47-56 页。斯普林格出版社，柏林。

Koza, J. R. and Rice, J. P. (1991). Genetic generation of both the weights and architecture for a neural network. In *Proceedings of International Joint Conference on Neural Networks, Seattle, July 1991.* Volume II, pp. 397-404. IEEE Press.

科扎和赖斯，J.p. (1991)。神经网络的权重和结构的遗传生成。1991 年 7 月，西雅图，神经网络国际联席会议会议记录。第二卷，第 397-404 页。IEEE 出版社。

Koza, J. R. and Rice, J. P. (1992a). *Genetic Programming: The Movie.* MIT Press, Cambridge, MA.

科扎，J.r。和赖斯，J.p。(1992a)。《基因编程: 电影》，麻省理工学院出版社，马萨诸塞州剑桥。

Koza, J. R. and Rice, J. P. (1992b). Automatic programming of robots using genetic programming. In *Proceedings of Tenth National Conference on Artificial Intelligence,* pp. 194 201. AAAI Press/MIT Press, Menlo Park, CA.

(1992 年 b)。使用遗传编程的机器人自动编程。在第十届全国人工智能会议的会议记录中，第 194201 页。AAAI 出版社/麻省理工学院出版社，加利福尼亚州门洛公园。

Koza, J. R., Keane, M. A. and Rice, J. P. (1993). Performance improvement of machine learning via automatic discovery

Koza，J.r。，Keane，M.a。 and Rice，J.p。(1993)通过自动发现提高机器学习的性能

112 Koza

of facilitating functions as applied to a problem of symbolic system identification. In *1993 IEEE International Conference on Neural Networks, San Francisco,* Volume I, pp. 191-198. IEEE Press, Piscataway, NJ.

应用于符号系统辨识问题的简化函数。1993 年 IEEE 国际神经网络会议，旧金山，第一卷，第 191-198 页。IEEE 出版社，皮斯卡塔韦，新泽西。

Langton, C., Taylor, C., Farmer, J. D. and Rasmussen, S. (EDS). (1992). *Artificial Life II, SFI Studies in the Sciences of Complexity,* Volume X. Addison-Wesley, Redwood City, CA.

兰顿、泰勒、法默和拉斯穆森(EDS)。(1992).人工生命 II，复杂性科学研究，卷十。艾迪生-韦斯利，雷德伍德城，加利福尼亚州。

Meyer, J.-A. and Wilson, S. W. (1991). *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior, Paris. September 24-28, 1990.*

迈耶，j-a。和威尔逊(1991)。从动物到动物：巴黎第一届模拟适应性行为国际会议记录。1990 年 9 月 24-28 日。

MIT Press, Cambridge, MA.

麻省理工学院出版社，马萨诸塞州剑桥。

Michaelewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York.

Michaelewicz，z。(1992)。遗传算法 + 数据结构 = 进化程序。Springer-Verlag，纽约。

Rawlins, G. (ED) (1991). *Proceedings of Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Bloomington, Indiana, July 15-18, 1990.* Morgan Kaufmann, San Mateo, CA.

罗林斯女士(教育署)(1991)。1990 年 7 月 15 日至 18 日，印第安纳州 Bloom-ington，关于遗传算法和分类器系统的研讨会会议记录。摩根 ·考夫曼，圣马特奥，加利福尼亚州。

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IB M Journal of Research and Development,* 3, 210-229.

塞缪尔，A.l. (1959)。一些关于使用跳棋游戏的机器学习的研究。研究与发展期刊，3,210-229。

Schaffer, J. D. (ED) (1989). *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA.

第三届国际遗传算法会议记录摩根 ·考夫曼，圣马特奥，加利福尼亚州。

Schwefel, H.-P. and Maenner, R. (EDS) (1991). *Parallel Problem Solving from Nature.* Springer-Verlag, Berlin.

(EDS)(1991)。《自然》中的并行问题解决。

Smith, S. F. (1980). *A Learning System Based on Genetic Adaptive Algorithms.* PhD dissertation, University of Pittsburgh, Pittsburgh, PA.

《基于遗传自适应算法的学习系统》，匹兹堡大学博士论文，宾夕法尼亚州匹兹堡。

Whitley, D. (~D) (1992). *Proceedings of Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Vail, Colorado 1992.* Morgan Kaufmann, San Marco, CA.

(~ d)(1992).遗传算法和分类器系统研讨会会议记录，韦尔，科罗拉多州 1992。摩根 ·考夫曼，圣马可，加利福尼亚州。

Wilson, S. W. (1987a). Classifier systems and the animat problem.
分类器系统和动画问题。

Machine Learning, 3, 199-228.

机器学习，3,199-228。

Wilson, S. W. (1987b). Hierarchical credit allocation in a classifier system. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* pp. 217-220. Morgan Kaufmann, San Mateo, CA.

威尔逊，西南部(1987b)。分类器系统中的分层信用分配。第十届国际人工智能联合会议记录，第 217-220 页。摩根 ·考夫曼，圣马特奥，加利福尼亚州。

Wilson, S. W. (1988). Bid competition and specificity reconsidered. *Jounal of Complex Systems,* 2, 705-723.

西南威尔逊(1988)。出价竞争和特异性反思。复杂系统杂志，2,705-723。