Титульник

Оглавление

[1. Introduction and Overview. Введение и обзор 2](#_Toc421084204)

[2. Introduction to Genetic Algorithms. Введение в генетические алгоритмы 4](#_Toc421084205)

[2.1. Why the Genetic Algorithm Works. Почему генетический алгоритм работает 7](#_Toc421084206)

[3. The Representation Problem for Genetic Algorithms. Проблема представления в генетических алгоритмах 8](#_Toc421084207)

[3.1. Previous Work 9](#_Toc421084208)

[3.2. Introduction to Lisp 9](#_Toc421084209)

[4. Overview of Genetic Programming. Обзор генетического алгоритма 10](#_Toc421084210)

[5. Detailed Description of Genetic Programming. Подробное описание генетического программирования 18](#_Toc421084211)

[5.1. The Structures Undergoing Adaptation. Адаптируемые структуры 18](#_Toc421084212)

[5.1.1. Closure of the Function Set and Terminal Set. Замкнутость функционального и терминального множеств 21](#_Toc421084213)

[5.1.2. Sufficiency of the Function Set and the Terminal Set. Достаточность функционального и терминального множеств 24](#_Toc421084214)

[5.1.3. Universality of Selecting Primitive Functions and Terminals. Универсальность выбора примитивных функций и терминалов 24](#_Toc421084215)

[5.2. The Initial Structures. Начальные структуры 25](#_Toc421084216)

[5.3. Fitness. Пригодность 33](#_Toc421084217)

[5.3.1. Raw Fitness. Исходный фитнес 35](#_Toc421084218)

[5.3.2. Standardized Fitness 36](#_Toc421084219)

[5.3.3. Adjusted Fitness 37](#_Toc421084220)

[5.3.4. Normalized Fitness 37](#_Toc421084221)

[5.4. Primary Operations for Modifying Structures 37](#_Toc421084222)

[5.4.1. Reproduction 37](#_Toc421084223)

[5.4.2. Crossover 37](#_Toc421084224)

[5.5. Secondary Operations 38](#_Toc421084225)

[5.5.1. Mutation 38](#_Toc421084226)

[5.5.2. Permutation 39](#_Toc421084227)

[5.5.3. Editing 39](#_Toc421084228)

[5.5.4. Encapsulation 39](#_Toc421084229)

[5.5.5. Decimation 39](#_Toc421084230)

# Introduction and Overview. Введение и обзор

In nature, biological structures that are more successful in grappling (борются) with their environment survive and reproduce at a higher rate (скорость). Biologists interpret the structures they observe in nature as the consequence of Darwinian natural selection operating in an environment over a period of time. In other words, in nature, structure is the consequence of fitness. Fitness causes, over a period of time, the creation of structure via natural selection and the creative effects of sexual recombination (genetic crossover) and mutation. That is, fitness begets structure.

В природе биологические структуры, которые наиболее успешно борются со своей окружающей средой, выживают и размножаются с более высокой скоростью. Биологи понимают структуры, которые они наблюдают в природе, как следствие естественного отбора по Дарвину, действующего в среде в течение промежутка времени. Другими словами, в природе структура – это следствие пригодности. То есть пригодность порождает структуру, основываясь на естественном отборе, а также с помощью половой рекомбинации (генетического скрещивания) и мутации.

Symbolic regression (i.e., function identification) involves finding a mathematical expression, in symbolic form, that provides a good, best, or perfect fit between a given finite sampling of values of the independent variables and the associated values of the dependent variables. That is, symbolic regression involves finding a model that fits a given sample of data.

Символьная регрессия заключается в нахождении математического выражения, в символьной форме, которое обеспечивает хорошее, лучшее или совершенное соответствие между заданной конечной выборкой значений независимых переменных и связанной выборкой значений зависимых переменных. То есть, символьная регрессии заключается в нахождении модели, которая соответствует заданной выборке данных.

When the variables are real-valued, symbolic regression involves finding both the functional form and the numeric coefficients for the model. Symbolic regression differs from conventional linear, quadratic, or polynomial regression, which merely involve finding the numeric coefficients for a function whose form (linear, quadratic, or polynomial) has been prespecified.

Когда переменные являются вещественными числами, символьная регрессия включает в себя одновременно и поиск вида функции, и поиск числовых коэффициентов модели. Символьная регрессия отличается от традиционной линейной, квадратичной или полиномиальной регрессии, которые просто находят числовые коэффициенты для функции, вид которой заранее известен.

In any case, the mathematical expression being sought in symbolic function identification can be viewed as a computer program that takes the values of the independent variables as input and produces the values of the dependent variables as output.

В любом случае, поиск математического выражения в символьной форме может рассматриваться в качестве компьютерной программы, которая принимает значения независимых переменных в качестве входных параметров и вычисляет значения зависимых переменных на выходе.

In the case of noisy data from the real world, this problem of finding the model from the data is often called empirical discovery. If the independent variable ranges over the non-negative integers, symbolic regression is often called sequence induction (as described above). Learning of the Boolean multiplexer function (also called Boolean concept learning) is symbolic regression applied to a Boolean function. If there are multiple dependent variables, the process is called symbolic multiple regression.

В случае зашумленности данных проблему нахождения функциональной зависимости данных часто называют эмпирическим открытием. Если независимые переменные находятся в промежутке неотрицательных чисел, то символьную регрессию часто называют последовательной индукцией. Если есть несколько зависимых переменных, процесс называется символьной множественной регрессией

# Introduction to Genetic Algorithms. Введение в генетические алгоритмы

In nature, the evolutionary process occurs when the following four conditions are satisfied: • An entity has the ability to reproduce itself. • There is a population of such self-reproducing entities. • There is some variety among the self-reproducing entities. • Some difference in ability to survive in the environment is associated with the variety

В природе эволюция происходит при наличии следующих условий:

* Организм способен воспроизводить себя;
* Существует популяция таких способных к размножению особей;
* Есть некоторое разнообразие организмов;
* Некоторые различия в способности выжить связаны с этим разнообразием

In nature, variety is manifested as variation in the chromosomes of the entities in the population. This variation is translated into variation in both the structure and the behavior of the entities in their environment. Variation in structure and behavior is, in turn, reflected by differences in the rate of survival and reproduction. Entities that are better able to perform tasks in their environment (i.e., fitter individuals) survive and reproduce at a higher rate; less fit entities survive and reproduce, if at all, at a lower rate. This is the concept of survival of the fittest and natural selection described by Charles Darwin in On the Origin of Species by Means of Natural Selection (1859). Over a period of time and many generations, the population as a whole comes to contain more individuals whose chromosomes are translated into structures and behaviors that enable those individuals to better perform their tasks in their environment and to survive and reproduce. Thus, over time, the structure of individuals in the population changes because of natural selection. When we see these visible and measurable differences in structure that arose from differences in fitness, we say that the population has evolved. In this process, structure arises from fitness.

В природе разнообразие обеспечивается различием хромосом особей популяции. Это различие преобразуется в изменение структуры организма и его поведения в среде. А это в свою очередь влияет на способность выживания в среде и скорость размножения. Организмы, которые способны лучше выполнять задачи в своей среде, чаще выживают и чаще размножаются, в отличие от менее пригодных особей. Эта концепция естественного отбора и выживания сильнейших была описана Чарльзом Дарвином в книге «О происхождении видов путем естественного отбора» (1859). С течением времени это приводит к тому, что в популяции остаются только особи с такими структурами организма и поведением, которые позволяют им выживать и производить себе подобных. Таким образом, структура особей в популяции меняется из-за естественного отбора. Когда мы видим ощутимые различия в структуре, возникшие из-за разницы в пригодности особей, то говорим, что популяция эволюционировала.

When we have a population of entities, the existence of some variability having some differential effect on the rate of survivability is almost inevitable. Thus, in practice, the presence of the first of the above four conditions (self-reproducibility) is the crucial condition for starting the evolutionary process

Когда у нас есть популяция особей, то наличие различий, дифференцированно влияющих на способность выжить, почти неизбежно. Поэтому на практике достаточно только первого условия для начала эволюции.

John Holland's pioneering book Adaptation in Natural and Artificial Systems (1975) provided a general framework for viewing all adaptive systems (whether natural or artificial) and then showed how the evolutionary process can be applied to artificial systems. Any problem in adaptation can generally be formulated in genetic terms. Once formulated in those terms, such a problem can often be solved by what we now call the "genetic algorithm."

Новаторская книга Джона Холланда «Adaptation in Natural and Artificial Systems» (1975) дала основу для наблюдения за всеми адаптивными системами, а затем показала, как эволюционный процесс может быть применен к искусственным системам. Любая проблема адаптации может быть сформулирована в генетических терминах. А после формулировки такая проблема может быть решена генетическим алгоритмом.

The genetic algorithm simulates Darwinian evolutionary processes and naturally occurring genetic operations on chromosomes. In nature, chromosomes are character strings in nature's base-4 alphabet. The four nucleotide bases that appear along the length of the DNA molecule are adenine (A), cytosine (C), guanine (G), and thymine (T). This sequence of nucleotide bases constitutes the chromosome string or the genome of a biological individual. For example, the human genome contains about 2,870,000,000 nucleotide bases.

Генетический алгоритм симулирует эволюционные процессы Дарвина и природные генетические операции с хромосомами.

The genetic algorithm is a highly parallel mathematical algorithm that transforms a set (population) of individual mathematical objects (typically fixed-length character strings patterned after chromosome strings), each with an associated fitness value, into a new population (i.e., the next generation) using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations (notably sexual recombination).

Генетический алгоритм является параллельным генетическим алгоритмом, который преобразует набор отдельных математических объектов (как правило, символьных строк фиксированной длины), каждый из которых связан с соответствующим значением фитнес-функции (пригодности), в новую популяцию (следующее поколение) с помощью операций, основанных на концепции выживания сильнейших по Дарвину и природных генетических операций (в частности, половой рекомбинации).

## Why the Genetic Algorithm Works. Почему генетический алгоритм работает

It would superficially appear that testing the four random strings does nothing more than provide values of fitness for those four explicitly tested points.

# The Representation Problem for Genetic Algorithms. Проблема представления в генетических алгоритмах

Representation is a key issue in genetic algorithm work because genetic algorithms directly manipulate a coded representation of the problem and because the representation scheme can severely limit the window by which a system observes its world. The conventional genetic algorithm operating on fixed-length character strings is capable of solving a great many problems. The mathematical tractability of fixed- length character strings (as compared with mathematical structures which are more complex) permitted Holland and subsequent researchers to construct a significant body of theory as to why genetic algorithms work. Nonetheless, the use of fixed-length character strings leaves many issues unsettled.

Представление является ключевым вопросом в генетическом алгоритме, потому что генетические алгоритмы работаю непосредственно с кодовым представлением проблемы. Обычный генетический алгоритм, работающий с символьными строками фиксированной длины, способен решить множество проблем. Тем не менее, использованием строк заданной длины оставляет много вопросов нерешенными.

For many problems, the most natural representation for a solution is a hierarchical computer program rather than a fixed-length character string. The size and the shape of the hierarchical computer program that will solve a given problem are generally not known in advance, so the program should have the potential of changing its size and shape. It is difficult, unnatural, and constraining to represent hierarchical computer programs of dynamically varying sizes and shapes with fixed-length character strings.

Для большинства проблем наиболее естественным представлением для решения проблемы является иерархическая компьютерная программа, а не символьная строка заданной длины. Размер и вид иерархической компьютерной программы, которая позволит решить данную проблему, как правило, неизвестны заранее, поэтому программа должна иметь возможность изменения размера и вида. Представлять символьные строки фиксированной длины в виде иерархической компьютерной программы, способной динамически изменять размер и вид, довольно трудно и неестественно.

Representation schemes based on fixed-length character strings do not readily provide the hierarchical structure central to the organization of computer programs (into programs and subroutines) and the organization of behavior (into tasks and subtasks).

Representation schemes based on fixed-length character strings do not provide any convenient way of representing arbitrary computational procedures or of incorporating iteration or recursion when these capabilities are desirable or necessary to solve a problem.

Moreover, such representation schemes do not have dynamic variability. The initial selection of string length limits in advance the number of internal states of the system and limits what the system can learn.

The predetermination of the size and shape of solutions and the pre-identification of the particular components of solutions has been a bane of machine learning systems from the earliest times (Samuel 1959).

## Previous Work

One approach to the problem of representation in genetic algorithms has been to provide greater flexibility by increasing the complexity of the structures undergoing adaptation in the genetic algorithm.

## Introduction to Lisp

As will be seen, the genetic programming paradigm described in this book applies many of the key ideas of the conventional genetic algorithm to structures that are more complex than character strings patterned after chromosome strings and considerably more general and expressive than the specialized structures used in past work on extending the conventional genetic algorithm. In particular, genetic programming operates with very general, hierarchical computer programs.

The term "computer program," of course, carries the connotation of the ability to do more than merely perform compositions of simple arithmetic operations. Among the connotations of the term "computer program" is the ability to perform alternative computations conditioned on the outcome of intermediate calculations, to perform operations in a hierarchical way, and to perform computations on variables of many different types.

# Overview of Genetic Programming. Обзор генетического алгоритма

The genetic programming paradigm 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 structures undergoing adaptation in genetic programming are general, hierarchical computer programs of dynamically varying size and shape.

Генетическое программирование пробует решить проблему представления в генетических алгоритмах путем увеличения сложности адаптируемых структур. В частности, адаптируемые структуры в генетическом программировании являются общими иерархическими компьютерными программами, динамически изменяющими размер и вид.

Many seemingly different problems 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.

Многие, казалось бы, разные проблемы в искусственном интеллекте, символьной обработке и машинном обучении можно рассматривать как требующие компьютерной программы, вычисляющей некоторый требуемый результат в зависимости от входных параметров.

I claim that 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 of 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.

Процесс решения этих проблем можно сформулировать как поиск наиболее подходящей индивидуальной компьютерной программы среди всех возможных компьютерных программ. Пространство поиска состоит из всех возможных компьютерных программ, составленных из функций и терминальных символов, соответствующих проблемной области. Генетическое программирование предоставляет способ поиска этих наиболее подходящих индивидуальных компьютерных программ.

In genetic programming, populations of hundreds or thousands of computer programs are genetically bred. This breeding is done using the Darwinian principle of survival 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 problem may emerge from this combination of Darwinian natural selection and genetic operations.

В генетическом программировании популяции сотен или тысяч компьютерных программ генетически выведены. Эта селекция осуществляется с помощью принципа выживания сильнейших и воспроизводства наиболее приспособленных особей вместе с генетической рекомбинацией (скрещиванием) организмов путем применения операций, подходящих для компьютерных программ. Компьютерная программа, которая решает (или приблизительно решает) определенную проблему может возникнуть из комбинации естественного отбора Дарвина и генетических операций.

Genetic programming starts with an initial population of randomly generated computer programs composed of functions and terminals appropriate to the problem domain. The functions may be standard arithmetic operations, standard programming operations, standard mathematical functions, logical functions, or domain-specific functions. 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 random population is, in effect, a blind random search of the search space of the problem

Генетическое программирование начинается генерации случайным образом начальной популяции компьютерных программ из функций и терминалов, соответствующих проблемной области. Функции могут быть стандартными арифметическими операциями, операциями программирования, математическими функциями, логическими функциями или предметно-ориентированными фикциями. В зависимости от конкретной задачи, компьютерная программа может работать с логическими значениями, целыми, вещественными или комплексными числами, векторами, символами. Создание этой начальной популяции в действительности «слепой» случайный поиск в пространстве проблемной области

Each individual computer program in the population is measured in terms of how well it performs in the particular problem environment. This measure is called the fitness measure. The nature of the fitness measure varies with the problem.

Каждая программа в популяции оценивается, насколько хорошо она выполняет свои задачи в проблемной среде. Эта оценка носит название фитнес-меры (меры пригодности). Ее вид зависит от проблемы.

For many 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. For some problems, fitness may be consist of a combination of factors such as correctness, parsimony, or efficiency.

Для многих задач пригодность естественно измерять ошибкой, погрешностью компьютерной программы. Чем ближе эта ошибка к нулю, тем лучше данная программа. Также может фитнес может быть комбинацией таких факторов, как корректность, экономность и бережливость.

Typically, each computer program in the population is run over a number of different fitness cases so that its fitness is measured as a sum or an average over a variety of representative different situations. These fitness cases sometimes represent a sampling of different values of an independent variable or a sampling of different initial conditions of a system. For example, the fitness of an individual computer program in the population may be measured in terms of the sum of the absolute value of the differences between the output produced by the program and the correct answer to the problem. This sum may be taken over a sampling of 50 different inputs to the program. The 50 fitness cases may be chosen at random or may be structured in some way.

Как правило, каждая компьютерная программа популяции отработает в нескольких фитнес случаях, тогда фитнес будет считаться в виде суммы или среднего арифметического значений фитнеса всех случаев. Эти фитнес случаи могут быть выборкой из различных значений независимой переменной или представлять собой выборку из различных начальных условий системы. Например, пригодность компьютерной программы может быть суммой абсолютной величины от разности вычисленного программой значения и корректного решения проблемы. Эта сумма может быть получена из выборки 50 различных входных значений программы. 50 фитнес случаев могут быть выбраны случайным образом или структурированы другим способом.

Unless the problem is so small and simple that it can be easily solved by blind random search, the computer programs in generation 0 will have exceedingly poor fitness. Nonetheless, some individuals in the population will turn out to be somewhat fitter than others. These differences in performance are then exploited.

За исключением случаев, когда проблема мала и проста, она не может быть легко решена путем слепого случайного поиска, компьютерные программы нулевого поколения будут иметь очень плохой фитнес. Тем не менее, некоторые особи в популяции будут немного пригоднее остальных. Эти различия в эффективности следует использовать в дальнейшем.

The Darwinian principle of reproduction and survival of the fittest and the genetic operation of sexual recombination (crossover) are used to create a new offspring population of individual computer programs from the current population of programs.

Принципы репродукции и выживания наиболее приспособленных особей и генетические операции половой рекомбинации (скрещивание) используются для создания нового поколения индивидуальных компьютерных программ из текущей популяции.

The reproduction operation involves selecting, in proportion to fitness, a computer program from the current population of programs, and allowing it to survive by copying it into the new population.

Операция репродукции включает в себя селекцию, пропорциональную значениям фитнеса, компьютерных программ из текущей популяции и позволяет отобранным особям выжить путем копирования в новую популяцию.

The genetic process of sexual reproduction between two parental computer programs is used to create new offspring computer programs from two parental programs selected in proportion to fitness. The parental programs are typically of different sizes and shapes. The offspring programs are composed of subexpressions (subtrees, subprograms, subroutines, building blocks) from their parents. These offspring programs are typically of different sizes and shapes than their parents.

Генетический процесс полового скрещивания двух родителей – компьютерных программ – используется для создания новых потомков от родителей, выбранных пропорционально фитнесу. Программы-родители обычно имеют различный размер и форму. Программы-потомки составляются из подвыражений (поддеревьев, подпрограмм) родителей. Эти потомки, как правило, различаются размером и видом от своих родителей.

Intuitively, if two computer programs are somewhat effective in solving a problem, then some of their parts probably have some merit. By recombining randomly chosen parts of somewhat effective programs, we may produce new computer programs that are even fitter in solving the problem.

Интуитивно, если две компьютерные программы несколько эффективны в решении проблемы, то их части, возможно, тоже немного пригодны. Скрещивая случайно выбранные части относительно пригодных программ, мы можем получить новую компьютерную программу, которая даже лучше решает проблему.

After the operations of reproduction and crossover are performed on the current population, the population of offspring (i.e., the new generation) replaces the old population (i.e., the old generation).

После выполнения операций репродукции и скрещивания с текущей популяцией, популяция потомков (новое поколение) помещается в старую популяцию (прошлое поколение).

Each individual in the new population of computer programs is then measured for fitness, and the process is repeated over many generations.

Каждая особь новой популяции компьютерных программ затем проверяется на пригодность, и процесс повторяется в течение многих поколений.

At each stage of this highly parallel, locally controlled, 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.

На каждом этапе этого параллельного, локально управляемого, децентрализованного процесса состоянием процесса будет являться только текущая популяция особей. Движущая сила этого процесса состоит только в наблюдении за пригодностью особей текущей популяции.

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.

Данный алгоритм будет производить популяцию компьютерных программ, которые через много поколений, как правило, демонстрируют увеличение средней пригодности. Кроме того, это популяции могут быстро и эффективно приспосабливаться к изменениям в окружающей среде.

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.

Как правило, лучшая особь, которая появляется в любом по счету поколении, обозначается как результат генетического программирования.

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 behavior subsumes or suppresses another.

Важной особенностью генетического программирования является иерархический характер производимых программ. Результаты генетического программирования по своей природе, сути иерархичны.

The dynamic variability of the computer programs that are developed along the way to a solution is also an important 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.

Динамическая изменчивость также является важным признаком компьютерных программ, созданных генетическим программированием. Было бы трудно и неестественно пытаться заранее уточнить или ограничить размер и форму возможного решения. Более того, такая предварительная спецификация сужает пространство поиска решений и может исключить нахождение решения вообще.

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 programming consist of functions that are natural for the problem domain.

Еще одной важной особенностью генетического программирования выступает отсутствие или сравнительно малая роль предварительной обработки входных данных и постобработки выходных значений. Входные параметры, промежуточные результаты и выходные значения обычно выражаются непосредственно в терминах естественной терминологии предметной области. Элементы функционального множества также естественны для проблемной области.

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

И наконец, структуры, подвергающиеся адаптации, активны. Они не являются пассивными кодировками решения проблемы. Вместо этого с учетом компьютера, на котором происходит запуск, программы в генетическом программировании – это активные структуры, способные выполниться в их текущем виде.

The genetic programming paradigm is a domain-independent (weak) method. It provides a single, unified approach to the problem of finding a computer program to solve a problem.

Парадигма генетического программирования является независимой от проблемной области. Это обеспечивает единый, унифицированный подход к проблеме нахождения решения в виде компьютерной программы.

# Detailed Description of Genetic Programming. Подробное описание генетического программирования

Adaptation (or learning) involves the changing of some structure so that it performs better in its environment. Holland's Adaptation in Natural and Artificial Systems (1975) provides a general perspective on adaptation and identifies the key features common to all adaptive systems. In this chapter, we use this perspective to describe genetic programming in terms of the structures that undergo adaptation.

Адаптация (или обучение) предполагает изменение некоторой структуры так, чтобы она лучше выполняла свои задачи в окружающей среде.

## The Structures Undergoing Adaptation. Адаптируемые структуры

In every adaptive system or learning system, at least one structure is undergoing adaptation.

В каждой адаптивной системе или системе обучения, хотя бы одна структура подвергается адаптации.

For the conventional genetic algorithm and genetic programming, the structures undergoing adaptation are a population of individual points from the search space, rather than a single point. Genetic methods differ from most other search techniques in that they simultaneously involve a parallel search involving hundreds or thousands of points in the search space.

В обычном генетическом алгоритме и генетическом программировании в качестве адаптируемых структур выступает популяция особей из всего пространства поиска. Генетические метода отличаются от большинства других методов поиска тем, что они включают одновременный параллельный поиск с участием сотен или тысяч точек всего пространства поиска.

The individual structures that undergo adaptation in genetic programming are hierarchically structured computer programs. The size, the shape, and the contents of these computer programs can dynamically change during the process.

Отдельные адаптируемые структуры в генетическом программировании являются иерархически структурированными компьютерными программами. Размер, форма и содержание этих компьютерных программ может быть динамически изменено в ходе выполнения процесса.

The set of possible structures in genetic programming is the set of all possible compositions of functions that can be composed recursively from the set of Nfunc functions from F = {f1,f2,...,fNfunc} and the set of Nterm terminals from T = {a1,a2,...,aNterm}. Each particular function fi in the function set F takes a specified number z(fi) of arguments z(f1), z(f2) ..., z(fNfunc). That is, function fi has arity z(fi).

Множество возможных структур в генетическом программировании – это множество всех возможных композиций функций, которые могут быть составлены рекурсивно из функционального множества *F = {f1, f2, ..., fN\_func}* и множества терминальных символов *T = {a1, a2, ..., aN\_term}.* Каждая конкретная функция *fi* из функционального множества *F* принимает указанное число *z(fi)* аргументов. То есть функция *fi* имеет арность равную *z(fi)*.

The functions in the function set may include

arithmetic operations (+, -, \*, etc.),

mathematical functions (such as sin, cos, exp, and log),

Boolean operations (such as AND, OR, NOT),

conditional operators (such as If-Then-Else),

functions causing iteration (such as Do-Until),

functions causing recursion, and

any other domain-specific functions that may be defined.

Функционально множество может состоять из следующих элементов:

* арифметические операции (+, -, \* , /);
* математические функции (sin, cos, exp, log);
* логические операции (конъюнкция, дизъюнкция, отрицание);
* условные операторы (If-Then-Else);
* операторы циклов (for, while, do);
* другие проблемно-ориентированные функции.

The terminals are typically either variable atoms (representing, perhaps, the inputs, sensors, detectors, or state variables of some system) or constant atoms (such as the number 3 or the Boolean constant NIL). Occasionally, the terminals are functions taking no explicit arguments, the real functionality of such functions lying in their side effects on the state of the system (e.g., the artificial ant problem).

Терминальными символами обычно являются либо переменные «атомы» (представляющие входы, сенсоры, датчики или переменные состояния некоторой системы), либо постоянные «атомы» (такие как число 3 или логическая константа NIL). Иногда в качестве термов также берутся функции, не принимающие явных аргументов, реальная функциональность таких функций заключается в создании побочных эффектов для состояний системы.

The structures that undergo adaptation in genetic programming are different from the structures that undergo adaptation in the conventional genetic algorithm operating on strings. The structures that undergo adaptation in genetic programming are hierarchical structures. The structures that undergo adaptation in the conventional genetic algorithm are one-dimensional fixed-length linear strings. In Steven F. Smith's (1980, 1983) variation of the conventional genetic algorithm, the individual structures undergoing adaptation are one-dimensional linear variable length strings.

Адаптационные структуры в генетическом программировании отличаются от структур, подвергающихся адаптации, в обычном генетическом алгоритме, который оперирует строками. В обычном генетическом алгоритме структуры представляют одномерные линейные строки фиксированной длины. В вариации Стивена Смита (1980б 1983) обычного генетического алгоритма адаптационные структуры являются одномерными линейными строками переменной длины.

In genetic programming, the terminal set and the function set should be selected so as to satisfy the requirements of closure and sufficiency.

В генетическом программировании терминальное и функциональное множества должны быть выбраны так, чтобы они удовлетворяли требованиями замкнутости и достаточности.

### Closure of the Function Set and Terminal Set. Замкнутость функционального и терминального множеств

The closure property requires that each of the functions in the function set be able to accept, as its arguments, any value and data type that may possibly be returned by any function in the function set and any value and data type that may possibly be assumed by any terminal in the terminal set. That is, each function in the function set should be well defined and closed for any combination of arguments that it may encounter.

Свойство замкнутости требует, чтобы каждая функция из функционального множества могла принять в качестве аргумента любое значение и тип данных, которые могут быть возвращены любой функцией из функционального множества, а также любой элемент терминального множества. То есть каждая функция из функционального множества должна быть четко определена и замкнута для любой комбинации аргументов, с которыми она может встретиться.

In ordinary programs, arithmetic operations operating on numerical variables are sometimes undefined (e.g., division by zero). Many common mathematical functions operating on numerical variables are also sometimes undefined (e.g., logarithm of zero). In addition, the value returned by many common mathematical functions operating on numerical variables is sometimes a data type that is unacceptable in a particular program (e.g., square root or logarithm of a negative number). Moreover, the Boolean value (i.e., T or NIL) typically returned by a conditional operator is generally not acceptable as the argument to an ordinary arithmetic operation.

В обычных программах арифметические операции с численными переменными иногда не определены (например, деление на нуль). Многие обычные математические функции иногда не определены (например, логарифм нуля). Кроме того, возвращаемые некоторыми математическими функциями значения могут входить в список неприемлемых типов данных для проблемной области (например, квадратный корень или логарифм отрицательного числа). Также логическое значение, обычно возвращаемое условным оператором, как правило, не принимается в качестве аргумента арифметическими функциями.

It therefore might appear that satisfaction of this closure property is not possible for ordinary computer programs, or that if possible, it would call for a very complex and restrictive syntactic structure to be imposed on the programs. In fact, as we will see, this is not the case. Closure can be achieved in a straightforward way for the vast majority of problems merely by careful handling of a small number of situations.

Может показаться, что соблюдение свойства замкнутости для обычной компьютерной программы невозможно или приведет к очень сложной и ограниченной синтаксической структуре. На самом деле это не так. Замкнутость может быть достигнута простым способом для подавляющего большинства задач просто путем тщательной обработки небольшого количества ситуаций.

If the arithmetic operation of division can encounter the numerical value of 0 as its second argument, the closure property will not be satisfied unless some arrangement is made to deal with the possibility of division by 0. One simple approach to guarantee closure is to define a protected division function. The protected division function % takes two arguments and returns one when division by 0 is attempted (including 0 divided by 0), and, otherwise, returns the normal quotient.

Если арифметическая операция деления получит в качестве второго аргумента число 0, то свойство замкнутости будет нарушено. Один простой подход гарантирует замкнутость – определение защищенной функции деления. Защищенная функция деления принимает два аргумента и возвращает 1 при попытке деления на 0 (включая деление 0 на 0), а в других случаях возвращает нормальное частное.

The closure property is desirable, but it is not absolutely required. If this closure property does not prevail, we must then address alternatives such as discarding individuals that do not evaluate to an acceptable result or assigning some penalty to such infeasible individuals. The issue of how to handle infeasible points is not unique to genetic methods and has been extensively (and inconclusively) debated in connection with numerous other algorithmic methods. There is no entirely satisfactory general resolution of this issue, so all the examples in this book will satisfy the closure property and we do not address this issue further.

Свойство замкнутости желаемо, но не абсолютно необходимо. Если свойство замкнутости не превалирует, то существуют альтернативные способы: исключение особей с нежелательным результатом или система штрафов для таких структур. Вопрос обработки таких ситуаций не уникален для генетических методов, а широко обсуждается в связи с другими алгоритмами. Удовлетворительного решения этой проблемы пока не существует, поэтому мы будет соблюдать свойство замкнутости.

### Sufficiency of the ***Function Set and the Terminal Set. Достаточность функционального и терминального*** множеств

The sufficiency property requires that the set of terminals and the set of primitive functions be capable of expressing a solution to the problem. The user of genetic programming should know or believe that some composition of the functions and terminals he supplies can yield a solution to the problem.

Свойство достаточности требует, чтобы терминальное и функциональное множества могли выразить решение проблемы. Пользователь генетического программирования должен убедиться, что композиция функция и термов приведет к решению проблемы.

The step of identifying the variables that have sufficient explanatory power to solve a particular problem is common to virtually every problem in science.

Этап определения переменных, у которых достаточно возможностей решить определенную проблему, является общим практически для каждой проблемы в науке.

Depending on the problem, this identification step may be obvious or may require considerable insight.

В зависимости от проблемы этот шаг идентификации может быть очевидным, а может потребовать глубокого понимания предметной области.

### Universality of Selecting Primitive Functions and Terminals. Универсальность выбора примитивных функций и терминалов

The steps (performed by the user) of determining the repertoire of primitive functions and terminals in genetic programming are equivalent to similar required steps in other machine learning paradigms. These two steps (which often go under other names) are often not explicitly identified, discussed, or recognized by researchers describing other paradigms. The reason for this omission may be that the researcher involved considers the choice of primitive functions and terminals to be inherent in the statement of the problem. This view is especially understandable if the researcher is focusing on only one specific type of problem from one specific field. If this book contained only one problem from only one field (e.g., only the artificial ant problem), it probably would not occur to the reader to think about the source of the primitive functions being used by the machine learning paradigm.

Этапы определения примитивных функций и терминальных символов в генетическом программировании эквивалентны аналогичным необходимым этапам в других парадигмах машинного обучения. Эти два шага часто явно не определяются, обсуждаются или признаются исследователями других парадигм. Причиной этого упущения может быть то, что исследователь считает выбор примитивных функций и терминалов присущим формулировке задачи. Такой взгляд особенно понятен, если ученый фокусируется лишь на одном конкретном типе проблемы специфической области.

## The Initial Structures. Начальные структуры

The initial structures in genetic programming consist of the individuals in the initial population of individual S-expressions for the problem.

Начальные структуры в генетическом программировании состоят из особей исходной популяции, каждая из которых представляет решение проблемы в виде польской записи.

The generation of each individual S-expression in the initial population is done by randomly generating a rooted, point-labeled tree with ordered branches representing the S-expression.

Создание каждого выражения начальной популяции выполняется в виде дерева со случайно выбранным корнем и упорядоченными ветвями, представляющего данную польскую нотацию.

We begin by selecting one of the functions from the set F at random (using a uniform random probability distribution) to be the label for the root of the tree. We restrict the selection of the label for the root of the tree to the function set F because we want to generate a hierarchical structure, not a degenerate structure consisting of a single terminal.

Начинаем со случайного выбора одной функции из функционального множества *F*, которая станет корнем дерева. Мы ограничиваем выбор функциональным множеством, поскольку нам необходимо создать иерархическую структуру, а не вырожденную структуру, состоящую из одного терминального символа.

Figure 6.2 shows the beginning of the creation of a random program tree. The function + (taking two arguments) was selected from a function set F as the label for the root of the tree.

Рисунок 1 демонстрирует начало создания случайного дерева программы. Функция + (с двумя аргументами) была выбрана случайно из функционального множества *F* в качества корня дерева.



Рисунок 1-Создание корня дерева

Whenever a point of the tree is labeled with a function f from F, then z(f) lines, where z(f) is the number of arguments taken by the function f, are created to radiate out from that point. Then, for each such radiating line, an element from the combined set C = F È T of functions and terminals is randomly selected to be the label for the endpoint of that radiating line.

Когда узел дерева помечается функцией *f* из *F*, то *z(f)* линий, где *z(f)* – количество аргументов функции *f*, выходит из этого узла. Затем для каждой такой линии случайно выбирается элемент из объединенного множества , функций и терминалов, для конечной точки – другого узла – этой линий.

If a function is chosen to be the label for any such endpoint, the generating process then continues recursively as just described above. For example, in figure 6.3, the function \* from the combined set C = F È T of functions and terminals was selected as the label of the internal nonroot point (point 2) at the end of the first (leftmost) line radiating from the point with the function + (point 1). Since a function was selected for point 2, it will be an internal, nonroot point of the tree that will eventually be created. The function \* takes two arguments, so the figure shows two lines radiating out from point 2.

Если в качестве узла была выбрана функция, то дальнейшее создание дерева продолжается рекурсивно так, как было описано выше. Например, на рисунке 2, функция \* была выбрана из множества в качестве внутреннего узла (номер 2) для конечной точки левой линии корневого узла (функция +, номер 1). Функция \* принимает два аргумента, поэтому из 2-ой вершины выходят две линии.



Рисунок 2-Выбор внутреннего узла

If a terminal is chosen to be the label for any point, that point becomes an endpoint of the tree and the generating process is terminated for that point. For example, in figure 6.4, the terminal A from the terminal set T was selected to be the label of the first line radiating from the point labeled with the function \*. Similarly, the terminals B and C were selected to be the labels of the two other radiating lines in figure 6.3. This process continues recursively from left to right until a completely labeled tree has been created, as shown in figure 6.4.

Если в качестве любого узла выбирается терминальный символ, то этот узел становится листом, и процесс создания поддерева для этой вершины прекращается. Например, на рисунке 3 терминальный символ A был выбран в качестве вершины для левой линий функции \*. Аналогичным образом, терминалы B и C стали вершинами двух правых линий функций \* и + соответственно. Этот процесс продолжается рекурсивно слева направо, пока все дерево не создано, как показано на рисунке 3.



Рисунок 3-Созданное дерево программы

This generative process can be implemented in several different ways resulting in initial random trees of different sizes and shapes. Two of the basic ways are called the ''full'' method and the "grow" method. The depth of a tree is defined as the length of the longest nonbacktracking path from the root to an endpoint.

Данный процесс генерации может быть реализован различными способами, приводящими к получению случайных начальных деревьев разного размера и вида. Два основных метода называются «полным» методом и «растущим» методом. Длина дерева определяется, как длина самого длинного пути от корня к листу.

The "full" method of generating the initial random population involves creating trees for which the length of every nonbacktracking path between an endpoint and the root is equal to the specified maximum depth. This is accomplished by restricting the selection of the label for points at depths less than the maximum to the function set F, and then restricting the selection of the label for points at the maximum depth to the terminal set T.

Полный метод создания начальной популяции состоит в генерации дерева, у которого длина каждого пути от листа к корню равна указанной максимальной глубине. Такой вид дерева можно получить путем ограничения выбора функциональным множеством для вершин, у которых глубина меньше заданной. А затем ограничить выбор только терминальным множеством для вершин с максимальной глубиной.

The "grow" method of generating the initial random population involves growing trees that are variably shaped. The length of a path between an endpoint and the root is no greater than the specified maximum depth. This is accomplished by making the random selection of the label for points at depths less than the maximum from the combined set C = F È T consisting of the union of the function set F and the terminal set T, while restricting the random selection of the label for points at the maximum depth to the terminal set T. The relative number of functions in the function set F and the number of terminals in the terminal set T determine the expected length of paths between the root and the endpoints of the tree.

Растущий метод генерации начальной популяции состоит в создании деревьев различной формы. Длина пути от листа до корня не больше заданной максимальной глубины. Это достигается путем случайного выбора внутреннего узла из множества , при этом длина пути от корня к узлу меньше максимальной глубины. А вершины дерева, чья длина равна максимальной, становятся листами и выбираются из терминального множества.

The generative method that I believe does best over a broad range of problems is a method I call "ramped half-and-half." In genetic programming, we usually do not know (or do not wish to specify) the size and shape of the solution in advance. The ramped half-and-half generative method produces a wide variety of trees of various sizes and shapes.

Лучший результат для широкого диапазона проблемы дает объединенный метод (ramped half-and-half), при котором начальная популяция создается чередованием полного и растущего методов. В генетическом программировании мы, как правило, не знаем заранее (или не хотим указывать) размер и форму решения. Объединенный метод генерации деревьев создает широкое разнообразие деревьев различного размера и вида.

The "ramped half-and-half" generative method is a mixed method that incorporates both the full method and the grow method. I have now adopted this method for all new problems and it is used for most problems in this book. The exceptions are the special analysis of Boolean functions in chapter 9 and a few runs made before my adoption of this method. The ramped half-and-half generative method involves creating an equal number of trees using a depth parameter that ranges between 2 and the maximum specified depth. For example, if the maximum specified depth is 6 (the default value in this book), 20% of the trees will have depth 2, 20% will have depth 3, and so forth up to depth 6. Then, for each value of depth, 50% of the trees are created via the full method and 50% of the trees are produced via the grow method.

Объединенный метод генерации – это смешанный метод, включающий и растущий, и полный методы. Данный способ состоит из создания равного количества деревьев с глубиной, которая находится в интервале от 2 до максимальной заданной глубины. Например, если максимальная глубина равна 6, то 20% деревьев будут иметь глубину 2, 20% деревьев глубину 3, и так далее до глубины 6. Затем для каждого значения глубины 50% деревьев создаются полным методом, а остальные 50% растущим методом.

Note that, for the trees created with the full method for a given depth, all paths from the root of the tree to an endpoint are the same length and therefore have the same shape. In contrast, for the trees created via the grow method for a given value of depth, no path from the root of the tree to an endpoint has a depth greater than the given value of depth. Therefore, for a given value of depth, these trees vary considerably in shape from one another.

Заметим, что у всех деревьев, созданных полным методом с заданной глубиной, длина пути от корня к листу одинаковая, равная максимальной глубине, и поэтому эти деревья имеют одинаковую форму. В отличие от этого, у всех деревьев, полученных растущим методом с данным значением глубины, ни один путь от корня дерева до листа не превышает максимальной глубины. Поэтому эти деревья значительно отличаются друг от друга по виду даже при одинаковой максимальной глубине.

Thus, the ramped half-and-half method creates trees having a wide variety of sizes and shapes.

Таким образом, объединенный метод создает деревья с большим разнообразием размеров и видов.

Duplicate individuals in the initial random generation are unproductive deadwood; they waste computational resources and undesirably reduce the genetic diversity of the population. Thus, it is desirable, but not necessary, to avoid duplicates in the initial random population. In genetic programming, duplicate random individuals are especially likely to be created in the initial random generation when the trees are small (as it is for a certain percentage of population in the ramped half-and-half and grow methods). Thus, each newly created S-expression is checked for uniqueness before it is inserted into the initial population. If a new S-expression is a duplicate, the generating process is repeated until a unique S-expression is created. Occasionally (e.g., for small trees), we must substitute a larger tree during the generative process when we have exhausted the set of possible trees of a given size.

Повторяющиеся особи в начальной популяции непродуктивны: они тратят вычислительные ресурсы и приводят к нежелательному сокращению генетического разнообразия популяции. Поэтому желательно, но необязательно, предотвратить появление дубликатов в начальной случайно популяции. В генетическом программировании вероятность появления повторяющихся особей в начальной популяции особенно высока, когда деревья малы. Таким образом, каждое созданное выражение проверяется на уникальность перед добавлением в популяцию. Если новое выражение – дубликат, то процесс повторяется до создания уникальной особи. Иногда (например, для маленьких деревьев) мы должны подставить большее по размеру дерево во время процесса генерации, если исчерпано множество возможных деревьев данного размера.

The variety of a population is the percentage of individuals for which no exact duplicate exists elsewhere in the population. If duplicate checking is done, the variety of the initial random population is 100%. In later generations, the creation of duplicate individuals via the genetic operation of reproduction is an inherent part of genetic processes.

Разнообразие популяции – это доля особей, у которых нет точной копии во всей популяции. Если выполняется проверка на дублирование при создании особей, то разнообразие начальной популяции равно 100%. В последующих поколениях появление одинаковых особей при использовании репродукции является неотъемлемой частью генетического процесса.

In contrast, in the conventional genetic algorithm operating on fixed-length character strings, each of the characters in a string in the initial random population is typically created by calling a binary randomizer. For example, the binary strings of length 453 used by Jefferson et al. (1991) in the artificial ant problem are created by a binary randomizer and come from a search space of size 2453 (i.e., about 10137). It would be most unusual to have any duplicates among the mere 65,536 individual strings in the population when the search space is of size 10137. Thus, in conventional genetic algorithms, no effort is usually expended to ensure against duplicates. However, duplicate checking is sometimes done (Davis 1991).

В противоположность этому в обычном генетическом алгоритме, работающем со строками символом фиксированной длины, каждый из символов строки начальной популяции обычно создается с помощью применения бинарного генератора случайных чисел. Например, бинарные строки длиной 453, используемые Джефферсоном и другими (1991), создаются с помощью двоичного генератора случайных чисел, имеющего пространство поиска размером 2453 (т.е. около 10137). Было бы необычно получить дубликаты среди всего лишь 65,536 индивидуальных строк популяции, когда пространство поиска размером 10137. Поэтому в обычных генетических алгоритмах, как правило, не проводится проверка одинаковых особей.

## Fitness. Пригодность

Fitness is the driving force of Darwinian natural selection and, likewise, of both conventional genetic algorithms and genetic programming.

Фитнес является движущей силой естественного отбора по Дарвину, а также обычных генетических алгоритмов и генетического программирования.

In nature, the fitness of an individual is the probability that it survives to the age of reproduction and reproduces. This measure may be weighted to consider the number of offspring. In the artificial world of mathematical algorithms, we measure fitness in some way and then use this measurement to control the application of the operations that modify the structures in our artificial population.

В природе пригодность – это вероятность того, что особь доживет до репродукционного возраста и воспроизведется. Данный показатель может учитываться при расчете числа потомков. В искусственном мире математических алгоритмов мы оцениваем пригодность каким-либо способом, а затем используем фитнес для контроля применения операций, изменяющих структуры в нашей искусственной популяции.

Fitness may be measured in many different ways, some explicit and some implicit.

Фитнес может быть вычислен при помощи различных методов, явных и неявных.

The most common approach to measuring fitness is to create an explicit fitness measure for each individual in the population. This approach is used in the vast majority of applications of the conventional genetic algorithm and for the vast majority of examples in this book. Each individual in a population is assigned a scalar fitness value by means of some well-defined explicit evaluative procedure.

Наиболее распространенным подходом вычисления пригодности является создание определенной фитнес меры для каждой особи популяции. Данный подход используется подавляющим большинством обычных генетических алгоритмов. Каждой особи популяции присваивается скалярное значение пригодности при помощи некоторой четко и явно определенной процедуры оценки.

Fitness may also be computed in a co-evolutionary way as when the fitness of a game playing strategy is determined by playing that strategy against an entire population (or sampling) of opposing strategies.

Фитнес также может быть вычислен путем совместной эволюции (коэволюции), при которой пригодность игровой стратегии определяется применением это стратегии против всей популяции (или отобранного числа) противоположной стратегии.

The fact that individuals exist and survive in the population and successfully reproduce may be indicative of their fitness (as is the case in nature). This implicit definition of fitness is often used in research in artificial life (Ray 1990, 1991a, 1991b, 1991c; Holland 1990, 1992; chapter 28 below). However, for the moment, we will focus on the more common situation where fitness is explicitly computed. I will now describe the four measures of fitness that are used in this book:

raw fitness,

standardized fitness,

adjusted fitness, and

normalized fitness.

Тот факт, что особи существуют и выживают в популяции, а также успешно воспроизводятся, может свидетельствовать об их пригодности (как это происходит в природе). Такое неявное определение фитнеса часто используется в научных исследованиях (Ray 1990; Holland 1990, 1992). Однако, на данный момент, мы сосредоточимся на более общей ситуации, когда фитнес вычисляется явно.

Существует 4 меры фитнеса:

* исходный (raw) фитнес;
* стандартизованный (standardized) фитнес;
* отрегулированный (adjusted) фитнес;
* нормированный (normalized) фитнес.

### Raw Fitness. Исходный фитнес

Raw fitness is the measurement of fitness that is stated in the natural terminology of the problem itself. For example, raw fitness in the artificial ant problem was the number of pieces of food eaten by the ant. The more food, the better. Raw fitness ranged from 0 (i.e., the least food and therefore the worst value) to 89.

Исходный фитнес – это измерение фитнеса, сформулированное в естественной терминологии проблемы. Например, исходным фитнесом для символьной регрессии является погрешность полученного результата вычисления функции относительно желаемого. Чем меньше погрешность, тем лучше.

The most common definition of raw fitness used in this book is that raw fitness is error. That is, the raw fitness of an individual S-expression is the sum of the distances, taken over all the fitness cases, between the point in the range space returned by the S-expression for the set of arguments associated with the particular fitness case and the correct point in the range space associated with the particular fitness case. The Sexpression may be Boolean-valued, integer-valued, floating-point-valued, complex-valued, vector-valued, multiple-valued, or symbolic valued.

Наиболее общим определением исходного фитнеса является ошибка. То есть исходный фитнес отдельного выражения – это сумма расстояний для всех фитнес случаев между полученным результатов для конкретного фитнес случая и желаемым результатом для этого же фитнес случая. Выражение может быть логическим, целочисленным, вещественным, комплексным, вектором или символьным значением.

If the S-expression is integer-valued or floating-point-valued, the sum of distances is the sum of the absolute values of the differences between the numbers involved. When raw fitness is error, the raw fitness r(i, t) of an individual S-expression i in the population of size M at any generational time step t is

where S(i, j) is the value returned by S-expression i for fitness case j (of Ne cases) and where C(j) is the correct value for fitness case j.

Если выражение целочисленное или вещественное, то сумма дистанций вычисляется в виде суммы абсолютных значений разности полученного и необходимого результатов. Когда исходным фитнесом является ошибка, исходный фитнес r(i, t) отдельного выражения i в популяции размера M любого поколения шага t равна

*,*

где *S(i, j)* – вычисленное значение выражения i для фитнес случая j (всего N случаев), C(j) – необходимое значение для фитнес случая j.

### Standardized Fitness

The standardized fitness s(i, t) restates the raw fitness so that a lower numerical value is always a better value. For example, in an optimal control problem, one may be trying to minimize some cost measure, so a lesser value of raw fitness is better. Similarly, if, in a particular problem, one is trying to minimize error, a lesser value of raw fitness is better (and a raw fitness of 0 is best).

### Adjusted Fitness

In addition, for all problems in this book involving an explicit calculation of fitness, we apply an optional adjustment to fitness. The adjusted fitness measure a(i, t) is computed from the standardized fitness s(i, t) as follows:

### Normalized Fitness

If the method of selection employed is fitness proportionate (as is the case for all problems in this book except for the experiments with tournament selection found in section 25.7), the concept of normalized fitness is also needed.

### Primary Operations for Modifying Structures

This section describes the two primary operations used to modify the structures undergoing adaptation in genetic programming:

* Darwinian reproduction
* crossover (sexual recombination).

### Reproduction

The reproduction operation for genetic programming is the basic engine of Darwinian natural selection and survival of the fittest. The reproduction operation is asexual in that it operates on only one parental S-expression and produces only one offspring S-expression on each occasion when it is performed.

### Crossover

The crossover (sexual recombination) operation for genetic programming creates variation in the population by producing new offspring that consist of parts taken from each parent. The crossover operation starts with two parental S-expressions and produces two offspring S-expressions. That is, it is a sexual operation.

## Secondary Operations

In addition to the two primary genetic operations of reproduction and crossover in genetic programming, there are five optional secondary operations worth mentioning:

· mutation

· permutation

· editing

· encapsulation, and

· decimation.

### Mutation

The mutation operation introduces random changes in structures in the population.

In conventional genetic algorithms operating on strings, the mutation operation can be beneficial in reintroducing diversity in a population that may be tending to converge prematurely. In the conventional genetic algorithm, it is common for a particular symbol (i.e., an allele) appearing at a particular position on a chromosome string to disappear at an early stage of a run because that particular allele is associated with inferior performance, given the alleles prevailing at other positions of the chromosome string at that stage of the run. Then, because of the

nonlinearities of the problem, the now-extinct allele may be precisely what is needed to achieve optimal performance at a later stage of the run, since a different and better combination of alleles is now prevailing at the other positions of the chromosome string. The situation just described is not conjectural but is, in fact, very typical. Genetic methods are normally applied to problems with highly nonlinear search spaces, and this situation is the essence of what is involved in nonlinear search spaces.

### Permutation

The *permutation* operation is a generalization of the inversion operation for the conventional genetic algorithm operating on strings.

### Editing

The *editing* operation provides a means to edit and simplify S-expressions as genetic programming is running.

### Encapsulation

The encapsulation operation is a means for automatically identifying a potentially useful subtree and giving it a name so that it can be referenced and used later.

### Decimation

For some complex problems, the distribution of fitness values over the initial random population may be skewed so that a very large percentage of the individuals have very poor fitness (e.g., a raw fitness of 0). This skewing may occur in problems where individuals in the population are assigned some penalty value of fitness because they would otherwise consume an infinite amount of time (as in time optimal control problems or problems involving iterative loops). In such problems, enormous amounts of computer time may be expended and wasted in early generations on very poor individuals. Moreover, when a highly skewed distribution of fitness values occurs, the few individuals with marginally better fitness values immediately begin to dominate the population and the variety of the population quickly begins to drop. In genetic programming, the crossover operation is usually capable of quickly reintroducing variety into the population. However, because the selection of parents to participate in crossover is based on fitness, the crossover operation concentrates on the few individuals in the population with the marginally better fitness values.