Человеку свойственно использовать в качестве основы своих трудов природные явления, так, например, природа эволюции стала источником вдохновения для создания генетического кода, нашедшего широкое применение и использующегося для решения задач на графы, задач компоновки, составления расписаний, создания искусственного интеллекта и т.д. Следует отметить, что основные принципы генетических алгоритмов были хорошо описаны Холландом в его знаменитой работе «Adaptation in Natural and Artificial Systems» в 1975 году, где впервые был рассмотрен термин «генетический алгоритм».

Генетический алгоритм – это адаптивный метод поиска, который в последнее время часто используется для решения задач функциональной оптимизации. Сущность метода базируется на генетических процессах биологических организмов: биологические популяции развиваются в течение нескольких поколений, подчиняясь законам естественного отбора и по принципу "выживает наиболее приспособленный", открытому Чарльзом Дарвином. Подражая этому процессу генетические алгоритмы способны "развивать" решения реальных задач, если те соответствующим образом закодированы.

Генетические алгоритмы в различных формах применились ко многим научным и техническим проблемам и использовались при создании других вычислительных структур, например, автоматов или сетей сортировки. В машинном обучении они использовались при проектировании нейронных сетей или управлении роботами. Они также применялись при моделировании развития в различных предметных областях, включая биологические (экология, иммунология и популяционная генетика), социальные (такие как экономика и политические системы) и когнитивные системы.

Тем не менее, возможно наиболее популярное приложение генетических алгоритмов - оптимизация многопараметрических функций. Многие реальные задачи могут быть сформулированы как поиск оптимального значения, где значение - сложная функция, зависящая от некоторых входных параметров. В некоторых случаях, представляет интерес найти те значения параметров, при которых достигается наилучшее точное значение функции. В других случаях, точный оптимум не требуется - решением может считаться любое значение, которое лучше некоторой заданное величины. В этом случае, генетические алгоритмы - часто наиболее приемлемый метод для поиска "хороших" значений. Сила генетического алгоритма заключена в его способности манипулировать одновременно многими параметрами, эта особенность и использовалась в сотнях прикладных программ, включая проектирование самолетов, настройку параметров алгоритмов и поиск устойчивых состояний систем нелинейных дифференциальных уравнений.

Принцип работы генетического алгоритма выгодно отличается от большинства других алгоритмов оптимизации возможностью оперировать совокупностью особей (популяцией), которые представляют собой строки, кодирующие одно из решений задачи. С помощью функции приспособленности среди всех особей популяции выделяют:

* наиболее приспособленные (более подходящие решения), которые получают возможность скрещиваться и давать потомство;
* наихудшие (плохие решения), которые удаляются из популяции и не дают потомства.

Таким образом, приспособленность нового поколения в среднем выше предыдущего. Отметим, что в классическом генетическом алгоритме:

* начальная популяция формируется случайным образом;
* размер популяции (количество особей N) фиксируется и не

изменяется в течение работы всего алгоритма;

* каждая особь генерируется как случайная L-битная строка, где L-

длина кодировки особи;

* длина кодировки для всех особей одинакова.

Рассмотрим подробнее алгоритм работы генетического кода и его этапы (рисунок 1).

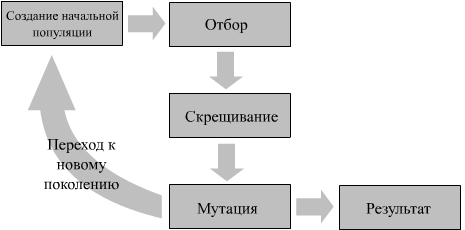


Рисунок 1 – Схема работы любого генетического алгоритма

Модель эволюционного процесса должна содержать популяцию (набор векторов), которая размножается, на которую воздействуют мутации и производится естественный отбор на основании минимизации целевой функции.

 На начальном этапе создается начальная популяция, которая, может оказаться не идеальной, однако велика вероятность, что алгоритм эту проблему исправит. Главное, чтобы она соответствовала «формату» и была приспособлена к размножению, основным принципом которого является — потомок похож на своих родителей. Т.е. необходимо задать какой-то механизм наследования. Желательно, чтобы он включал элемент случайности. Но скорость развития таких систем очень низкая — разнообразие генетическое падает, популяция вырождается. Для решения данной проблемы внедряются скрещивание особей промежуточной популяции путем кроссовера, что приводит к формированию нового поколения. На этапе скрещивания особи промежуточной популяции случайным образом разбиваются на пары, потом с некоторой вероятностью скрещиваются, в результате чего получаются два потомка, которые записываются в новое поколение, или не скрещиваются, тогда в новое поколение записывается сама пара. В классическом генетическом алгоритме применяется одноточечный оператор кроссовера: для родительских строк случайным образом выбирается точка раздела, потомки получаются путём обмена отсечёнными частями. Но существуют и альтернативы: двухточечный и равномерный кроссоверы. В двухточечном кроссовере выбираются две точки разрыва, и родительские хромосомы обмениваются сегментом, который находится между двумя этими точками. В равномерном кроссовере, каждый бит первого родителя наследуется первым потомком с заданной вероятностью; в противном случае этот бит передается второму потомку. И наоборот.

Далее происходит мутация нового поколения, необходимая для "выбивания" популяции из локального экстремума и способствующая защите от преждевременной сходимости. Каждый бит каждой особи популяции с некоторой вероятностью инвертируется. Эта вероятность обычно очень мала, менее 1%. Можно выбирать некоторое количество точек в хромосоме для инверсии, причем их число также может быть случайным. Также можно инвертировать сразу некоторую группу подряд идущих точек. Среди рекомендаций по выбору вероятности мутации нередко можно встретить варианты 1/L или 1/N.

При условии, что полученный результат нас не устраивает, эти шаги повторяются до тех пор, пока результат нас не начнет удовлетворять или произойдет одно из ниже перечисленных условий:

* Количество поколений (циклов) достигнет заранее выбранного максимума;
* Исчерпано время на мутацию.

Необходимо сказать, что во многих областях, ГД применяются генетические алгоритмы, необходимо чтобы они были непредсказуемы (например, криптография – генерация ключей). С этой целью американский институт по стандартам NIST разработал набор тестов для оценки случайности последовательности чисел, основанных на различных статистических свойствах, присущих только случайным последовательностям. Рассмотрим некоторые из них (таблица 1).

Таблица 1 - Статистические тесты NIST

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | частотный побитовый | monobit frequency | Цель — выяснить, действительно ли число нулей и единиц в последовательности приблизительно одинаковы, как это можно было бы предположить в случае истинно случайной бинарной последовательности. Тест оценивает, насколько близка доля единиц к 0,5. Если вычисленное в ходе теста значение вероятности *p* < 0,01, то данная двоичная последовательность не является истинно случайной. |
| 2 | Частотный блочный | block frequency | Цель — выяснить действительно ли частота повторения единиц в блоке длиной *m* бит приблизительно равна *m*/2, как можно было бы предположить в случае абсолютно случайной последовательности. Вычисленное в ходе теста значение вероятности *p* должно быть не меньше 0,01. В противном случае (*p* < 0,01) двоичная последовательность не носит истинно случайный характер. |
| 3 | Тест на последовательность одинаковых битов | runstest | Суть состоит в подсчёте полного числа рядов в исходной последовательности, где под словом «ряд» подразумевается непрерывная подпоследовательность одинаковых битов. Ряд длиной *k* бит состоит из *k* абсолютно идентичных битов, начинается и заканчивается с бита, содержащего противоположное значение. Цель данного теста — сделать вывод о том, действительно ли количество рядов, состоящих из единиц и нулей с различными длинами, соответствует их количеству в случайной последовательности. В частности, определяется быстро либо медленно чередуются единицы и нули в исходной последовательности. Если вычисленное в ходе теста значение вероятности *p* < 0,01, то данная двоичная последовательность не является истинно случайной. |
| 4 | Тест на самую длинную последовательность единиц в блоке | longestrunones10000 | В данном тесте определяется самый длинный ряд единиц внутри блока длиной *m* бит. Цель — выяснить действительно ли длина такого ряда соответствует ожиданиям длины самого протяжённого ряда единиц в случае абсолютно случайной последовательности. Если высчитанное в ходе теста значение вероятности *p* < 0,01 полагается, что исходная последовательность не является случайной. |
| 5 | Тест рангов бинарных матриц | binarymatrixranktest | Здесь производится расчёт [рангов](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%BD%D0%B3_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B) непересекающихся подматриц, построенных из исходной двоичной последовательности. Целью этого теста является проверка на линейную зависимость подстрок фиксированной длины, составляющих первоначальную последовательность. В случае если вычисленное в ходе теста значение вероятности *p* < 0,01, делается вывод о неслучайном характере входной последовательности бит. |
| 6 | спектральный тест |  | Цель — выявление периодических свойств входной последовательности, например, близко расположенных друг к другу повторяющихся участков. Тем самым это явно демонстрирует отклонения от случайного характера исследуемой последовательности. Идея состоит в том, чтобы число пиков, превышающих пороговое значение в 95 % по амплитуде, было значительно больше 5 %. Если вычисленное в ходе теста значение вероятности *p* < 0,01, то данная двоичная последовательность не является абсолютно случайной. |
| 7 | Тест на совпадение неперекрывающихся шаблонов | nonoverlappingtemplatematchingtest | В данном тесте подсчитывается количество заранее определенных шаблонов, найденных в исходной последовательности. Цель — выявить генераторы случайных или псевдослучайных чисел, формирующие слишком часто заданные непериодические шаблоны. Используется окно также длиной *m* бит. Если шаблон не обнаружен, окно смещается на один бит. Если же шаблон найден, окно перемещается на бит, следующий за найденным шаблоном, и поиск продолжается дальше. Вычисленное в ходе теста значение вероятности *p* должно быть не меньше 0,01. |
| 8 | Тест на совпадение перекрывающихся шаблонов | overlappingtemplatematchingtest | Суть данного теста заключается в подсчете количества заранее определенных шаблонов, найденных в исходной последовательности. Как и в [тесте 7](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%8B_NIST#.D0.A2.D0.B5.D1.81.D1.82_.D0.BD.D0.B0_.D1.81.D0.BE.D0.B2.D0.BF.D0.B0.D0.B4.D0.B5.D0.BD.D0.B8.D0.B5_.D0.BD.D0.B5.D0.BF.D0.B5.D1.80.D0.B5.D0.BA.D1.80.D1.8B.D0.B2.D0.B0.D1.8E.D1.89.D0.B8.D1.85.D1.81.D1.8F_.D1.88.D0.B0.D0.B1.D0.BB.D0.BE.D0.BD.D0.BE.D0.B2) для поиска конкретных шаблонов длиной *m* бит используется окно также длиной *m* бит. Сам поиск производится аналогичным образом. Если шаблон не обнаружен, окно смещается на один бит. Разница между этим тестом и [тестом 7](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%8B_NIST#.D0.A2.D0.B5.D1.81.D1.82_.D0.BD.D0.B0_.D1.81.D0.BE.D0.B2.D0.BF.D0.B0.D0.B4.D0.B5.D0.BD.D0.B8.D0.B5_.D0.BD.D0.B5.D0.BF.D0.B5.D1.80.D0.B5.D0.BA.D1.80.D1.8B.D0.B2.D0.B0.D1.8E.D1.89.D0.B8.D1.85.D1.81.D1.8F_.D1.88.D0.B0.D0.B1.D0.BB.D0.BE.D0.BD.D0.BE.D0.B2) заключается лишь в том, что если шаблон найден, окно перемещается только на бит вперед, после чего поиск продолжается дальше. Вычисленное в ходе теста значение вероятности *p* должно быть не меньше 0,01. |
| 9 | Универсальный статистический тест Маурера | maurersuniversalstatistictest | Здесь определяется число бит между одинаковыми шаблонами в исходной последовательности (мера, имеющая непосредственное отношение к длине сжатой последовательности). Цель теста — выяснить может ли данная последовательность быть значительно сжата без потерь информации. В случае если это возможно сделать, то она не является истинно случайной. Если *p* < 0,01, то полагается, что исходная последовательность не является случайной. |
| 10 | Тест на линейную сложность | linearcomplexitytest | Цель — выяснить является ли входная последовательность достаточно сложной для того, чтобы считаться абсолютно случайной. Абсолютно случайные последовательности характеризуются длинными линейными регистрами сдвига с обратной связью. Если же такой регистр слишком короткий, то предполагается, что последовательность не является в полной мере случайной. Если *p* < 0,01, то полагается, что исходная последовательность не является случайной. |

Таким образом, хотя модель эволюционного развития, применяемая в генетических алгоритмах, сильно упрощена по сравнению со своим природным аналогом, тем не менее генетические алгоритмы являются достаточно мощным средством и могут с успехом применяться для широкого класса прикладных задач, включая те, которые трудно, а иногда и вовсе невозможно, решить другими методам. Однако, генетические алгоритмы, как и другие методы эволюционных вычислений, не гарантируют обнаружения глобального решения за полиномиальное время. Генетические алгоритмы не гарантируют и того, что глобальное решение будет найдено, но они хороши для поиска "достаточно хорошего" решения задачи "достаточно быстро". Там, где задача может быть решена специальными методам, почти всегда такие методы будут эффективнее генетического алгоритма и в быстродействии и в точности найденных решений. Главным же преимуществом генетических алгоритмов является то, что они могут применяться даже на сложных задачах, там, где не существует никаких специальных методов. Даже там, где хорошо работаю существующие методики, можно достигнуть улучшения сочетанием их с генетическими алгоритмами.