Генетические алгоритмы применяются для решения некоторых трудно или долго решаемых задач, или таких задач, для которых вообще не найдено решения.

Генетические алгоритмы в некоторой степени соприкасаются с нейронными сетями. Потому что и там и там идёт отбор удачных комбинаций, а для решения задачи алгоритм не должен её "понимать".

Как работает генетический алгоритм?

Он, как ни странно, работает в точности так, как настоящий естественный отбор (с некоторыми упрощениями, конечно).

Вместо того, чтобы решать сложную задачу, применяя формулы, теоремы и расчёты (это или получается слишком долго, или мы просто не знаем, как надо решать), мы выбираем случайные решения и надеемся на то, что они окажутся оптимальными. Но случайные решения можно перебирать очень долго, поэтому мы ускоряем процесс, отбирая те решения, которые более удачны, и строя новые решения на их основе.

То есть – имеется некий набор генов. Эти гены размножаются, мутируют, затем выживают наиболее приспособленные. Эти наиболее приспособленные гены затем опять размножаются и т.д.

Основной проблемой генетического алгоритма является определение, что такое ген, и критерий приспособленности (фитнес-функция).

1. Популяция хромосом

Если мы хотим вывести естественным отбором автомобиль, то для начала нужно описать гены автомобиля. Это могут быть: колесо, мотор, руль, рама и т.п.

Если мы хотим вывести часы, то генами часов могут быть: маятник, шестерёнки, циферблат, стрелки, пружина.

Далее эти гены каким-то образом соединяются между собой, образуя хромосому. Хромосома в нашем случае это то же самое что организм или особь, так как вся эволюция это выживание хромосом, а организм, тело, особь и т.п. это всего лишь защитный мобильный контейнер для хромосомы.

Например, у автомобиля колесо может присоединиться к рулю, или к раме, или к другому колесу. У часов шестеренка может присоединиться к циферблату, пружине или стрелке.

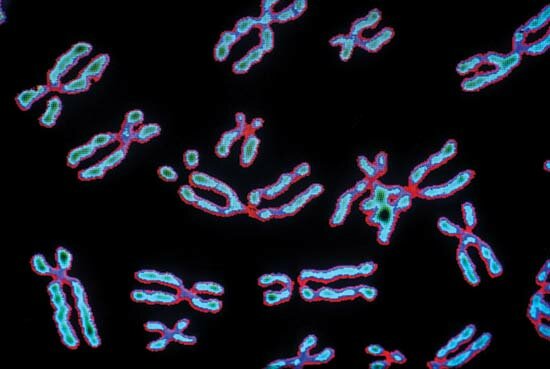
Подчеркну, что в случае моделирования каких-то физических объектов нужно делать именно физическую симуляцию взаимодействия их генов. Если же мы решаем некую абстрактную математическую задачу, то набор генов для неё это просто набор исходных переменных.

На первом шаге алгоритма создаётся популяция из некоторого количества организмов-хромосом. Каждая хромосома содержит случайный набор генов. Например, первичная популяция хромосом автомобиля:

Хромосома №1: "руль-руль-колесо-мотор-мотор-колесо-колесо"

Хромосома №2: "рама-рама-руль-колесо-колесо-мотор-мотор-руль"

и т.д.



2. Селекция

На этом этапе должна выжить часть наиболее приспособленных организмов-хромосом, а другая часть погибнуть.

Как определяется приспособленность? Это и есть главная проблема генетического алгоритма. Если мы выбрали хороший критерий приспособленности, алгоритм будет работать хорошо, а если плохой, то возможно алгоритм не будет работать вообще.

*Часто говорят так: если бы мы знали хороший критерий приспособленности, мы бы тогда решили задачу обычным способом, и генетический алгоритм был бы не нужен. Так что здесь не всё так уж радужно. Тем не менее, наблюдать за работой генетического алгоритма хотя бы интересно.*

Например, мы знаем, как выглядит функционально хороший автомобиль. Это 4 одинаковых колеса, расположенные в углах прямоугольной рамы, один руль, один мотор, и т.д. Зная этот критерий, мы могли бы отбирать все хромосомы, имеющие 4 одинаковых колеса, прямоугольную раму, и т.д.

Но очевидно, что применять генетический алгоритм здесь уже бессмысленно, так как мы уже знаем решение.

Вместо этого мы должны задать такие критерии, которые ничего не знают о строении автомобиля, но оценивают его качественно.

Таким критерием может быть: пройденное расстояние и затраченное время. То есть мы запускаем физическую симуляцию, и смотрим, кто из организмов смог вообще ехать, и кто проехал дальше всех. А уж из чего они там сделаны – не наше дело.

Для часов это может быть: угол поворота стрелки, который соответствует одному часу, допустим. Мы также с помощью симуляции проверям, у кого из организмов есть вообще стрелки, поворачиваются ли они, и с какой точностью.

Для математической задачи мы просто подставляем параметры в решение, и смотрим, удовлетворяет ли это решение нашим критериям (самый короткий путь, самая низкая цена и т.п.)

В соответствии с критерием приспособленности каждой хромосоме назначается рейтинг.

Затем, исходя из рейтинга, выбирается та часть хромосом, которая должна выжить. Выбор можно осуществлять несколькими методами. Например, есть метод элит и метод рулетки. Суть в том, что для разных задач более удачно работают разные методы, и их приходится подбирать. Но в целом, как бы ни назывался метод, его задача в том, чтобы выбрать наиболее приспособленные хромосомы. Скажем, метод элит просто отбирает хромосомы с самым высоким рейтингом.

3. Отбор

После того, как выжили самые приспособленные хромосомы (можно, например, оставить 50% популяции), среди них отбираются пары родителей для следующего поколения. Отбор тоже производится разными методами – панмиксия, инбридинг и т.д. Их мы рассмотрим позже, но несмотря на умные названия, ничего особенного в них нет. Как и всегда, наиболее приспособленные хромосомы имеют больше шансов размножиться и оставить потомство.

4. Скрещивание

После того, как кандидаты на размножение были отобраны, между ними происходит скрещивание и они создают потомство. Потомок может наследовать: гены первого родителя, гены второго родителя, гены обоих родителей, и также гены родителей и ещё кого-то. Для этого тоже есть разные методы: одноточечный кроссоверинг, триадный кроссоверинг и т.д. И опять же, эти методы не являются чем-то принципиально сложным и направлены лишь на то, чтобы добиваться лучшей эффективности алгоритма на разных задачах.

В результате скрещивания появляются потомки – новые хромосомы, набор генов у которых уже не случаен, а взят из наиболее успешных родителей. Что с большой вероятностью приведёт к тому, что эти потомки окажутся такими же приспособленными или ещё лучше.

После появления потомков популяция обновляется: родители уходят (или нет, это тоже вопрос выбора стратегии), потомки остаются.

5. Мутации

Генетические алгоритмы имеют склонность сходиться к "почти хорошей" хромосоме. Появившись в популяции, такая хромосома становится успешной, постоянно размножается, потомки наследуют её гены, и в скором времени вся популяция будет состоять только из этой хромосомы. Популяция застрянет на этом уровне развития, так и не достигнув по-настоящему хорошего решения.

Поэтому важную роль играют мутации. Они не дают популяции превратиться в набор одинаковых хромосом. (Вышеописанные разные методы селекции, отбора и скрещивания также направлены на решение этой проблемы.)

Мутации могут внедряться прямо во время скрещивания в некоторых методах. Но их также можно делать отдельным шагом. На этом шаге берётся некий процент хромосом, и в них случайным образом меняется некоторое количество генов. Сколько хромосом должно мутировать, и сколько генов нужно менять в каждой хромосоме – определяется опытным путём, чтобы не сломать алгоритм.

Характер мутаций зависит от того, как представлены гены. Например, для автомобиля мутация это изменение диаметра колеса, изменение размера рамы, изменение положения колеса и т.п.

Для часов это изменение силы пружины, изменение количества зубов в шестеренке, изменение длины маятника и т.п.

Для математических задач это просто замена одного числа на другое.

6. Повтор

Далее цикл повторяется. Снова определяются выжившие, затем выбираются родители, делаются потомки и т.д.

Цикл повторяется до тех пор, пока не будет найдено решение задачи, или пока не пройдёт заданное количество циклов.

Байтовые строки в Python - что это такое и с чем это едят? Байтовые строки очень похожи на [обычные строки](https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html), но с некоторыми отличиями. Попробуем выяснить, с какими.

Что такое байты? Байт - минимальная единица хранения и обработки цифровой информации. Последовательность байт представляет собой какую-либо информацию (текст, картинку, мелодию...).

Создаём байтовую строку:

>>>

**>>>** b'bytes'

b'bytes'

**>>>** 'Байты'.encode('utf-8')

b'\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'

**>>>** bytes('bytes', encoding = 'utf-8')

b'bytes'

**>>>** bytes([50, 100, 76, 72, 41])

b'2dLH)'

Если первые три способа нам уже известны ([тут](https://pythonworld.ru/tipy-dannyx-v-python/stroki-literaly-strok.html), [тут](https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html) и [тут](https://pythonworld.ru/osnovy/vstroennye-funkcii.html)), то последний нужно пояснить. Функция bytes принимает список чисел от 0 до 255 и возвращает байты, получающиеся применением функции chr.

>>>

**>>>** chr(50)

'2'

**>>>** chr(100)

'd'

**>>>** chr(76)

'L'

Что делать с байтами? Хотя байтовые строки поддерживают практически все строковые методы, с ними мало что нужно делать. Обычно их надо записать в [файл](https://pythonworld.ru/tipy-dannyx-v-python/fajly-rabota-s-fajlami.html) / прочесть из файла и преобразовать во что-либо другое (конечно, если очень хочется, то можно и распечатать). Для преобразования в строку используется метод decode:

>>>

**>>>** b'**\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b**'.decode('utf-8')

'Байты'

**Bytearray**

Bytearray в python - массив байт. От типа **bytes** отличается только тем, что является изменяемым. Про него, в общем-то, больше рассказать нечего.

>>>

**>>>** b = bytearray(b'hello world!')

**>>>** b

bytearray(b'hello world!')

**>>>** b[0]

104

**>>>** b[0] = b'h'

Traceback (most recent call last):

File "", line 1, in

b[0] = b'h'

TypeError: an integer is required

**>>>** b[0] = 105

**>>>** b

bytearray(b'iello world!')

**>>> for** i **in** range(len(b)):

**...**  b[i] += i

**...**

**>>>** b

bytearray(b'ifnos%}vzun,')