Генетические алгоритмы

Егоров Кирилл, гр. 3538
Чураков Михаил, гр. 3539

Оглавление:

1. Введение

Генетический алгоритм — это алгоритм, который позволяет найти удовлетворительное решение к аналитически неразрешимым или сложнорешаемым проблемам через последовательный подбор и комбинирование искомых параметров с использованием механизмов, напоминающих биологическую эволюцию.

1.1 Мягкие вычисления

Термин "мягкие вычисления" (Soft computing techniques) был введен Лофти Заде (основоположником нечёткой логики) в 1994 году. Это понятие объединяет такие области как:

- нечёткая логика
- нейронные сети
- вероятностные рассуждения
- сети доверия
- эволюционные вычисления

Они дополняют друг друга и используются в различных комбинациях или самостоятельно для создания гибридных интеллектуальных систем.

1.2 Эволюционные вычисления

Генетические алгоритмы являются частью более общей группы методов, называемой эволюционными вычислениями, которые объединяют различные варианты использования эволюционных принципов для достижения поставленной цели.

Также в ней выделяют следующие направления:

- Эволюционные стратегии
  - Метод оптимизации, основанный на идеях адаптации и эволюции. Степень мутации в данном случае меняется со временем – это приводит к, так называемой, самоадаптации.
- Генетическое программирование
  - Применение эволюционного подхода к популяции программ.
- Эволюционное программирование
  - Было впервые предложено Л.Дж. Фогелем в 1960 году для моделирования эволюции как процесса обучения с целью создания искусственного интеллекта. Он использовал конечные автоматы, предсказывающие символы в цифровых последовательностях, которые, эволюционируя, становились более приспособленными к решению поставленной задачи.

Генетические алгоритмы применяются для решения следующих задач:

- Оптимизация функций
- Разнообразные задачи на графах (задача коммивояжера, раскраска, нахождение паросочетаний)
- Настройка и обучение искусственной нейронной сети
- Задачи компоновки
- Составление расписаний
- Игровые стратегии
- Аппроксимация функций
- Искусственная жизнь
- Биоинформатика

2. История

2.1 Несколько открытий в биологии

Подоплекой возникновения этих методов стали несколько открытий в биологии.

Сначала Чарльз Дарвин опубликовал в 1859 году свою знаменитую работу "Происхождение видов", где были провозглашены основные принципы эволюционной теории:

- Наследственность (потомки сохраняют свойства родителей)
- Изменчивость (потомки почти всегда не идентичны)
- Естественный отбор (выживают наиболее приспособленные).

Тем самым было показано, какие принципы приводят к эволюционному развитию флоры и фауны под влиянием окружающей среды. Однако вопрос о том, как генетическая информация передается от родителей потомкам, долгое время оставался открытым.

В 1944 году Эйвери, Маклеод и Маккарти опубликовали результаты своих исследований, доказывавших, что за наследственные процессы ответственна "кислота дезоксирибозного типа". Однако о том, как работает ДНК, стало известно позднее.

В 1953 году в номере журнала "Nature" вышла статья Уотсона и Крика, которые впервые предложили модель двухцепочной спирали ДНК.

Таким образом, стали известны все необходимые компоненты для реализации эволюционной модели.

2.2 Ключевые работы

Первые публикация, которые можно отнести к генетическим алгоритмам, принадлежат Баричелли Н.А. Его работы "Symbiogenetic evolution processes realised by artificial methods" (1957), "Numerical testing of evolution theories" (1962) были направлены прежде всего на понимание природного феномена наследственности.

В 1966 году Л.Дж. Фогель, А.Дж. Оуэнс, М.Дж. Уолш предложили и исследовали эволюцию простых автоматов, предсказывающих символы в цифровых последовательностях.

Родителем современной теории генетических алгоритмов считается Д.Х. Холланд. Однако сначала его интересовала, прежде всего, способность природных систем к адаптации, а его мечтой было создание такой системы, которая могла бы приспосабливаться к любым условиям окружающей среды.

В 1975 году Холланд публикует свою самую знаменитую работу «Adaptation in Natural and Artificial Systems». В ней он впервые ввёл термин «генетический алгоритм» и предложил схему классического генетического алгоритма (canonical GA). В дальнейшем понятие «генетические алгоритмы» стало очень широким, и зачастую к ним относятся алгоритмы, сильно отличающиеся от классического ГА.

Ученики Холланда – Кеннет Де Йонг и Дэвид Голдберг – внесли огромный вклад в развитие ГА. Наиболее известная работа Голдберга – «Genetic algorithms in search optimization and machine learning» (1989).

3. Классический ГА

3.1 Постановка задачи и функция приспособленности

Пусть перед нами стоит задача оптимизации, например:

- Задача наилучшего приближения
  - Если рассматривать систему n линейных уравнений с m неизвестными

$$Ax = b$$

    в случае, когда она переопределена ($n > m$), то иногда оказывается естественной задача о нахождении вектора x, который "удовлетворяет этой системе наилучшим образом", т. е. из всех "не решений" является лучшим.

- Задача о рационе.
  - Пусть имеется n различных пищевых продуктов, содержащих m различных питательных веществ. Обозначим через a_{ij} содержание (долю) j-го питательного вещества в i-ом продукте, через b_j — суточную потребность организма в j-ом питательном веществе, через c_i — стоимость единицы i-го продукта. Требуется составить суточный рацион питания минимальной стоимости, удовлетворяющий потребность во всех питательных веществах
- Транспортная задача.
  - Эта задача — классическая задача линейного программирования. К ней сводятся многие оптимизационные задачи. Формулируется она так. На m складах находится груз, который нужно развезти n потребителям. Пусть a_i ($i = 1, \ldots, n$) — количество груза на i-ом складе, а b_j ($j = 1, \ldots, m$) — потребность в грузе j-го потребителя, c_{ij} — стоимость перевозки единицы груза с i-го склада j-му потребителю. Требуется минимизировать стоимость перевозок.
- Задачи о распределении ресурсов.
  - Общий смысл таких задач — распределить ограниченный ресурс между потребителями оптимальным образом. Рассмотрим простейший пример — *задачу о режиме работы энергосистемы.* Пусть m электростанций питают одну нагрузку мощности p. Обозначим через x_j активную мощность, генерируемую j-ой электростанцией. Техническими условиями определяются возможный минимум m_j и максимум M_j вырабатываемой j-ой электростанцией мощности. Допустим затраты на генерацию мощности x на j-ой электростанции равны $e_j(x)$. Требуется сгенерировать требуемую мощность p при минимальных затратах.

Переформулируем задачу оптимизации как задачу нахождения максимума некоторой функции $f(x_1, x_2, \ldots, x_n)$, называемой *функцией приспособленности* (fitness function). Она должна принимать неотрицательные значения на ограниченной области определения (для того, чтобы мы могли для каждой особи считать её приспособленность, которая не может быть отрицательной), при этом совершенно не требуются непрерывность и дифференцируемость.

Каждый параметр функции приспособленности кодируется строкой битов.

Особью будет называться строка, являющаяся конкатенацией строк упорядоченного набора параметров:

1010 10110 101 … 10101

| x1 | x2 | x3 | … | xn |

Универсальность ГА заключается в том, что от конкретной задачи зависят только такие параметры, как функция приспособленности и кодирование решений. Остальные шаги для всех задач производятся одинаково.

3.2 Принцип работы ГА

Генетические алгоритмы оперируют совокупностью особей (популяцией), которые представляют собой строки, кодирующие одно из решений задачи. Этим ГА отличается от большинства других алгоритмов оптимизации, которые оперируют лишь с одним решением, улучшая его.

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

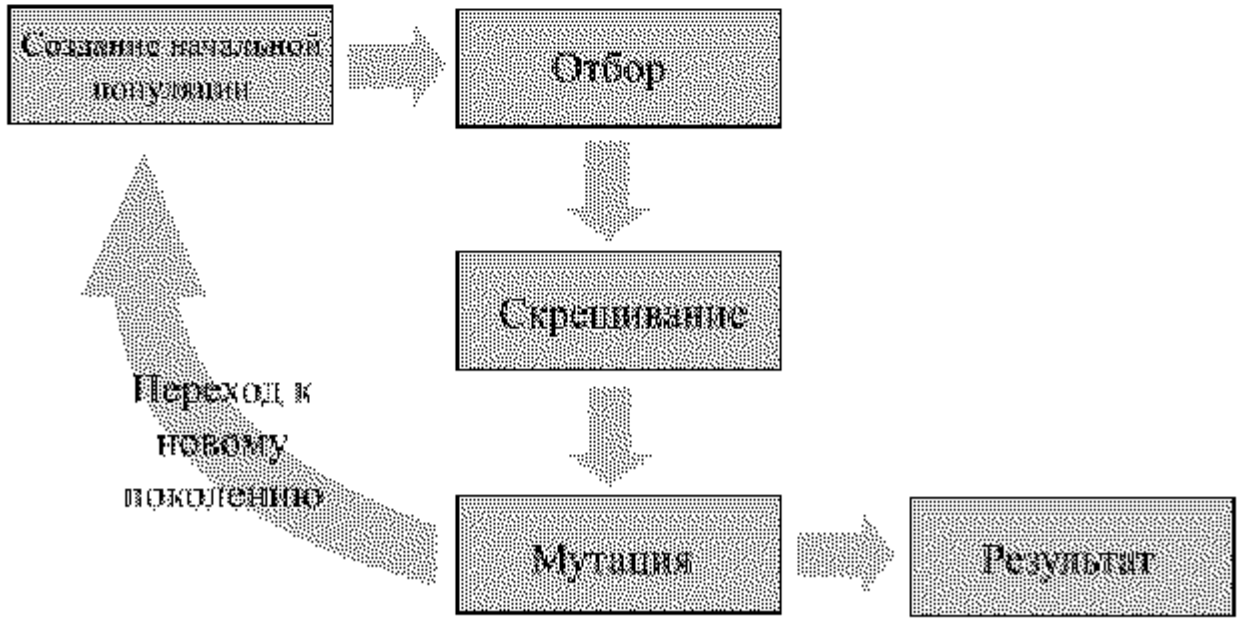
Таким образом, приспособленность нового поколения в среднем выше предыдущего.

В классическом ГА:

- 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

3.3 Алгоритм работы

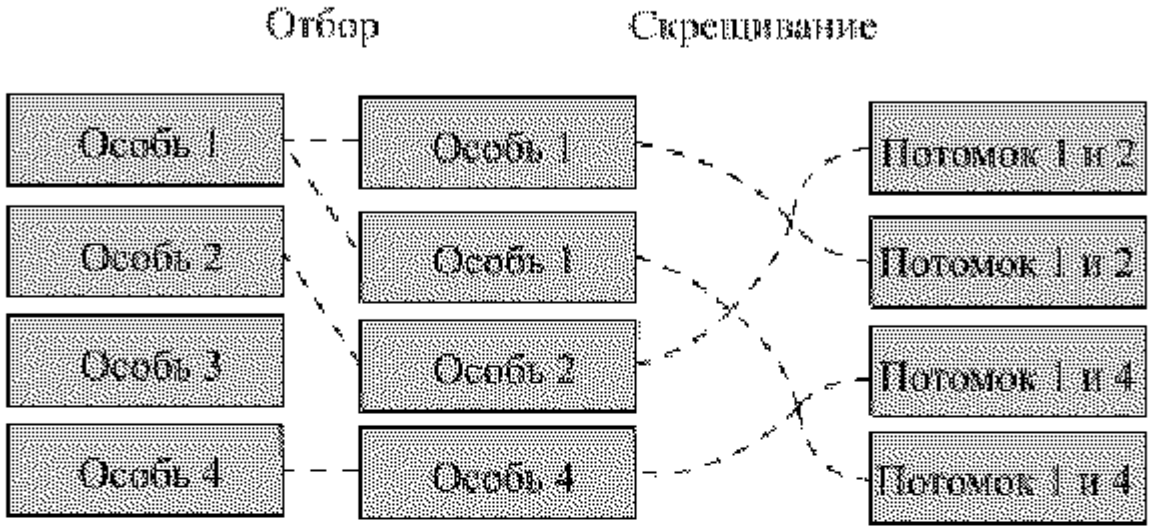
На рисунке изображена схема работы любого генетического алгоритма:



Шаг алгоритма состоит из трех стадий:

1. генерация промежуточной популяции (*intermediate generation*) путем отбора (*selection*) текущего поколения
2. скрещивание (*recombination*) особей промежуточной популяции путем *кроссовера* (*crossover*), что приводит к формированию нового поколения
3. мутация нового поколения

Первые две стадии (отбор и скрещивание):



3.4 Отбор

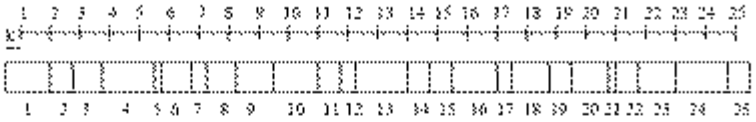
Промежуточная популяция — это набор особей, получивших право размножаться. Наиболее приспособленные особи могут быть записаны туда несколько раз, наименее приспособленные с большой вероятностью туда вообще не попадут.

В классическом ГА вероятность каждой особи попасть в промежуточную популяцию пропорциональна ее приспособленности, т.е. работает *пропорциональный отбор* (*proportional selection*).

Существует несколько способов реализации данного отбора:

- *stochastic sampling*. Пусть особи располагаются на колесе рулетки так, что размер сектора каждой особи пропорционален ее приспособленности. N раз запуская рулетку, выбираем требуемое количество особей для записи в промежуточную популяцию.
- *remainder stochastic sampling*. Для каждой особи вычисляется отношение ее приспособленности к средней приспособленности популяции. Целая часть этого отношения указывает, сколько раз нужно записать особь в промежуточную популяцию, а дробная показывает её вероятность попасть туда ещё раз. Реализовать такой способ отбора удобно следующим образом: расположим особи на рулетке так же, как было описано. Теперь пусть у рулетки не одна стрелка, а N, причем они отсекают одинаковые сектора. Тогда один запуск рулетки выберет

сразу все N особей, которые нужно записать в промежуточную популяцию. Такой способ иллюстрируется следующим рисунком:



3.5 Скрещивание

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 ГА применяется одноточечный оператор кроссовера (*1-point crossover*): для родительских строк случайным образом выбирается точка раздела, потомки получаются путём обмена отсечёнными частями.

```
011010.01010001101  ->  111100.01010001101
111100.10011101001      011010.10011101001
```

3.6 Мутация

К полученному в результате отбора и скрещивания новому поколению применяется оператор мутации, необходимый для "выбивания" популяции из локального экстремума и способствующий защите от преждевременной сходимости.

Каждый бит каждой особи популяции с некоторой вероятностью инвертируется. Эта вероятность обычно очень мала, менее 1%.

```
1011001100101101 -> 1011001101101101
```

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.

3.7 Критерии останова

Такой процесс эволюции, вообще говоря, может продолжаться до бесконечности. Критерием останова может служить заданное количество поколений или *схождение* (*convergence*) популяции.

Схождением называется состояние популяции, когда все строки популяции находятся в области некоторого экстремума и почти одинаковы. То есть кроссовер практически никак не изменяет популяции, а мутирующие особи склонны вымирать, так как менее приспособлены. Таким образом, схождение популяции означает, что достигнуто решение близкое к оптимальному.

Итоговым решением задачи может служить наиболее приспособленная особь последнего поколения.

4. Теория

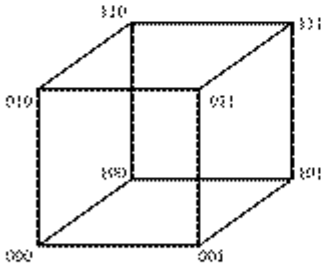
Рассмотрим несколько теоретических фактов, которые помогут более чётко понять природу генетических алгоритмов и примерно объяснить, почему же они работают.

4.1 Шаблоны

Шаблоном (*schema*) называется строка длины L из символов 0, 1 и *(«don't care» символ). Строка является представителем данного шаблона, если все символы кроме * совпадают. Например, у шаблона 1*0*0 есть 4 представителя:

- 10000
- 10010
- 11100
- 11110

Если представить пространство поиска в виде гиперкуба, то строки – это его вершины, а шаблон определяет в нем гиперплоскость. К примеру, шаблон *1* определяет верхнюю грань этого трехмерного куба:



Поэтому термины «шаблон» и «гиперплоскость» взаимозаменяемы.

Порядком шаблона называется количество фиксированных в нём битов.

Определяющей длиной шаблона называется расстояние между его крайними фиксированными битами.

Например, для шаблона H = *0**10*: порядок $o(H) = 3$, определяющая длина $\Delta(H) = 4$.

Очевидно, что количество представителей шаблона H равно $2^{L-o(H)}$, а количество шаблонов равно 3^L.

Приспособленностью шаблона называется средняя приспособленность строк из популяции, являющихся его представителями. Следует заметить, что эта величина зависит от популяции, и поэтому меняется со временем.

4.2 Неявный параллелизм

Внешне кажется, что генетический алгоритм при отборе выбирает строку, однако при этом неявным образом происходит выборка шаблонов, представителем которых она является. Это означает, что на каждом поколении количество представителей шаблона изменяется в соответствии с его текущей приспособленностью. У «хороших» шаблонов представители в среднем более приспособленные, а значит, они чаще будут выбираться в промежуточную популяцию. «Плохие» шаблоны имеют много шансов вымереть. Одна строка является представителем сразу многих шаблонов (а именно 2^L: на каждой позиции мы либо оставляем бит строки, либо заменяем его на «*»). Поэтому при отборе одной строки отбирается сразу целое множество шаблонов.

4.3 Теорема шаблонов

Теорема шаблонов (*The Schema Theorem*), приведённая Холландом, показывает, как изменяется доля представителей шаблона в популяции, являясь первой попыткой объяснить правильность работы генетических алгоритмов. Следует заметить, что она верна только для классического ГА с его пропорциональным отбором и одноточечным кроссовером.

$M(H, t)$ — число представителей шаблона H в поколении t.

Количество представителей шаблона H в промежуточном поколении:

$$M(H, t+intermediate) = M(H, t)\frac{f(H, t)}{< f(t) >}$$

где $f(H, t)$ — приспособленность шаблона H в поколении t, а $<f(t)>$ — средняя приспособленность поколения t.

Особи промежуточной популяции с вероятностью p_c подвергаются кроссоверу. Одноточечный кроссовер может разрушить шаблон – ни один из детей не является представителем рассматриваемого шаблона. Вероятность разрушения меньше, чем

$$\frac{\Delta(H)}{(L-1)}\left(1 - P(H, t)\frac{f(H, t)}{< f(t) >}\right)$$

где $P(H, t)$ — доля представителей шаблона H в поколении t. Первый множитель произведения равен вероятности попадания точки раздела между фиксированными битами шаблона, а второй — вероятности выбрать в пару представителя другого шаблона.

Но даже в случае, когда второй родитель не является представителем данного шаблона, и точка раздела попадает между фиксированными битами, шаблон не обязательно разрушается. Например, рассматриваем шаблон 11***, кроссоверу подвергаются строки 11011 и 10100, точка раздела попадает между первыми двумя битами

```
1.1011  ->  1.1011
1.0100      1.0100
```

Как видно, в этой ситуации шаблон не разрушается.

Переходя от количества представителей к их доле, получаем следующее неравенство:

$$P(H, t+1) \geq P(H, t) \frac{f(H, t)}{< f(t) >} \left(1 - p_c \frac{\Delta(H)}{(L-1)} \left(1 - P(H, t) \frac{f(H, t)}{< f(t) >} \right) \right)$$

Учтём влияние мутации. В шаблоне $o(H)$ фиксированных битов, и каждый не будет инвертирован с вероятностью $(1 - p_m)$. Откуда получаем **итоговую формулу теоремы шаблонов:**

$$P(H, t+1) \geq P(H, t) \frac{f(H, t)}{< f(t) >} \left(1 - p_c \frac{\Delta(H)}{(L-1)} \left(1 - P(H, t) \frac{f(H, t)}{< f(t) >} \right) \right) (1 - p_m)^{o(H)}$$

- Полученное выражение не слишком удачно для анализа работы генетического алгоритма, т.к. в нём присутствует знак неравенства.
- Мы не учитывали случаи, когда рассматриваемый шаблон получается в результате кроссовера пары строк, не являющихся его представителями.
- Приспособленность шаблона и средняя приспособленность популяции быстро изменяются от поколения к поколению, поэтому полученное неравенство хорошо описывает ситуацию только для следующего поколения.

На данный момент существуют более точные версии этой теоремы и другие рассуждения, доказывающие целесообразность использования генетических алгоритмов.

5. Настройка ГА

Генетический алгоритм производит поиск решений с помощью:

- отбора гиперплоскостей (*hyperplane sampling*), осуществляемого кроссовером, поскольку последний комбинирует и совмещает шаблоны родителей в их детях.
- метода *hill-climbing*, обеспечивающегося мутацией: особь случайным образом изменяется – неудачные варианты вымирают, полезные изменения сохраняются популяцией.

Исследования показали, что на простых задачах с малым размером популяции ГА с мутацией (и без кроссовера) находят решение быстрее. На сложных многоэкстремальных функциях лучше использовать ГА с кроссовером, поскольку этот метод более надежен, хотя и требует большего размера популяции.

С точки зрения теоремы шаблонов, мутация только вредит росту количества представителей хороших шаблонов, поскольку лишний раз их разрушает. Однако мутация просто необходима для ГА с малым размером популяции, потому что для них свойственна *преждевременная сходимость* (*premature convergence*) – ситуация, когда в некоторых позициях все особи имеют один и тот же бит, не соответствующий глобальному экстремуму.

Введём понятие, использующееся для анализа ГА. *Давление отбора* (*selection pressure*) — это мера того, насколько различаются шансы лучшей и худшей особей популяции попасть в промежуточную популяцию. Для пропорционального отбора эта величина с

увеличением средней приспособленности популяции уменьшается, стремясь к 1, т.е. шансы плохой и хорошей особей создать потомство уравниваются.

При увеличении вероятностей скрещивания или мутации и при уменьшении давления отбора (например, за счет использования других стратегий отбора) размножение представителей приспособленных шаблонов замедляется, но зато происходит интенсивный поиск других шаблонов. Обратно, уменьшение вероятностей скрещивания или мутации и увеличение давления отбора ведет к интенсивному использованию найденных хороших шаблонов, но меньше внимания уделяется поиску новых.

Вывод: для эффективной работы генетического алгоритма необходимо поддерживать тонкое равновесие между *исследованием и использованием*.

Необходимость *сбалансированной сходимости* ГА:

- быстрая сходимость может привести к схождению к неоптимальному решению
- медленная сходимость часто приводит к потере найденной наилучшей особи.

Методология управления сходимостью классического ГА до сих пор не выработана.

6. Различные модификации ГА

6.1 Кодирование

Аргументы в пользу кодирования бинарным алфавитом:

- обеспечивает лучший поиск с помощью гиперплоскостей, т. к. предоставляет максимальное их количество.

  Например, при кодировании 2^L значений для бинарного алфавита количество гиперплоскостей будет 3^L, а при использовании, четырехзначного алфавита – $5^{L/2}$.

- для встречаемости каждого символа в каждой позиции требуется меньший размер популяции

Даже для двух строк, есть вероятность, что на каждой позиции в популяции есть и 0, и 1. Если же алфавит большей мощности, то до применения мутации большая часть пространства поиска будет недоступна с точки зрения кроссовера, после применения мутации станет недоступна другая часть.

Однако небинарные алфавиты зачастую обеспечивают более наглядное представление решений задачи.

Для большинства функций ГА будут работать лучше при кодировании параметров кодом Грея, а не прямым бинарным кодом. Это связано с тем, что расстояние Хэмминга между битовыми представлениями данных может и не отражать близость в привычном смысле – например, числа 7 и 8 различаются на 4 бита. Бинарное кодирование добавляет дополнительные разрывы, что осложняет поиск.

Пример: пусть требуется минимизировать функцию $f(x) = x^2$

Если в начальной популяции преобладали хорошие отрицательные решения, то скорее всего мы придём к решению $-1 = 11\ldots1$. Но достигнуть глобального минимума $00\ldots0$ будет практически невозможно, поскольку изменение любого бита будет приводить к ухудшению решения. При кодировании кодом Грея такой проблемы не возникает.

Иногда применяется кодирование с плавающей точкой, которое тоже является более удачным, чем прямое бинарное, в силу того, что в некоторых случаях более правильно отражает понятие схожести параметров особей.

6.2 Стратегии отбора

Ранковый отбор (*rank selection*): для каждой особи ее вероятность попасть в промежуточную популяцию пропорциональна ее порядковому номеру в отсортированной по возрастанию приспособленности популяции. Такой вид отбора не зависит от средней приспособленности популяции.

Турнирный отбор (*tournament selection*): из популяции случайным образом выбирается t особей, и лучшая из них помещается в промежуточную популяцию. Этот процесс повторяется N раз, пока промежуточная популяция не будет заполнена. Наиболее распространен вариант при $t = 2$.

Отбор усечением (*truncation selection*): популяция сортируется по приспособленности, затем берется заданная доля лучших, и из них случайным образом N раз выбирается особь для дальнейшего развития.

6.3 Кроссовер

Двухточечный кроссовер: выбираются 2 точки раздела, и родители обмениваются промежутками между ними:

```
010.1001.1011  ->  010.1011.1011
110.1011.0100      110.1001.0100
```

При этом определяющая длина измеряется в кольце – для шаблона 1*****1 при двухточечном кроссовере она будет равна 1, хотя при одноточечном была 6.

Однородный кроссовер: один из детей наследует каждый бит с вероятностью p_0 у первого родителя и с $(1 - p_0)$ у второго, второй ребенок получает не унаследованные первым биты. Обычно $p_0 = 0.5$.

Однородный кроссовер в большинстве случаев разрушает шаблон, поэтому плохо предназначен для отбора гиперплоскостей, однако при малом размере популяции он препятствует преждевременному схождению.

6.4 Стратегии формирования нового поколения

Два основных типа формирования нового поколения после кроссовера и мутации:

- дети замещают родителей
- новое поколение составляется из совокупности и детей, и их родителей

Также применяется принцип *элитизма*: в новое поколение включается заданное количество лучших особей предыдущего поколения (часто одна лучшая особь).

Использование второй стратегии и элитизма не допускает потери лучших решений.

К примеру, если популяция сошлась в локальном максимуме, а мутация вывела одну из строк в область глобального, то при замещении родителей весьма вероятно, что эта особь в результате скрещивания будет потеряна, и решение задачи не будет получено. Если же используется элитизм, то полученное хорошее решение будет оставаться в популяции до тех пор, пока не будет найдено лучшее.

7. Некоторые модели ГА

7.1 Genitor (Whitley)

В данной модели используется специфичная стратегия отбора. На каждом шаге только *одна* пара случайных родителей создает только *одного* ребенка. Этот ребенок заменяет не родителя, а одну из худших особей популяции.

Таким образом, на каждом шаге в популяции обновляется только одна особь.

Исследования показали, что поиск гиперплоскостей происходит лучше, а сходимость быстрее, чем у классического ГА.

7.2 CHC (Eshelman)

CHC – это *Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation*.

Для нового поколения выбираются *N* лучших *различных* особей среди родителей и детей. Дублирование строк не допускается.

Для скрещивания все особи разбиваются на пары, но скрещиваются только те пары, между которыми расстояние Хэмминга больше некоторого порогового (также возможны ограничения на минимальное расстояние между крайними различающимися битами).

При скрещивании используется так называемый HUX-оператор (Half Uniform Crossover), разновидность однородного кроссовера – каждому потомку переходит ровно половина битов каждого родителя.

Размер популяции небольшой. Этим оправдано использование однородного кроссовера.

Данный алгоритм довольно быстро сходится из-за того, что в нем нет мутаций. В этом случае CHC применяет *cataclysmic mutation*: все строки, кроме самой приспособленной, подвергаются сильной мутации (изменяется около трети битов). Таким образом, алгоритм перезапускается и далее продолжает работу, применяя только кроссовер.

7.3 Hybrid algorithm (Davis)

Использование гибридного алгоритма позволяет объединить преимущества ГА с преимуществами классических методов.

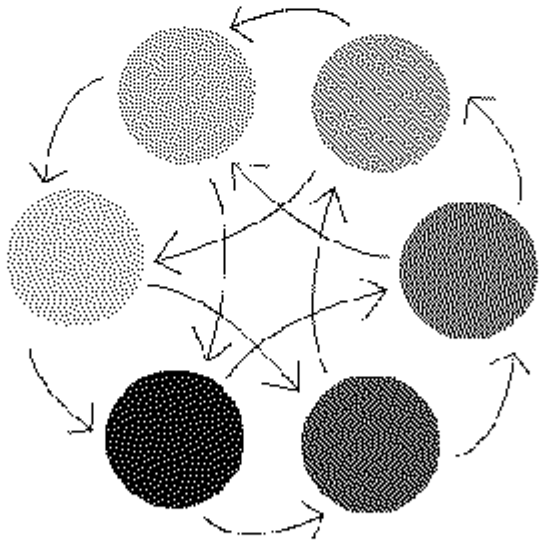
Дело в том, что ГА являются робастными алгоритмами, т.е. позволяют находить хорошее решение, но нахождение оптимального зачастую оказывается намного более трудной задачей в силу стохастичности принципов работы алгоритма. Поэтому возникла идея использовать ГА на начальном этапе для эффективного сужения пространства поиска вокруг глобального экстремума, а затем, взяв лучшую особь, применить один из "классических" методов оптимизации.

Однако можно использовать "классические" методы (*hill-climbing, например)* и внутри самих ГА. На каждом поколении каждый полученный потомок оптимизируется этим методом, таким образом, каждая особь достигает локального максимума, вблизи которого она находится, после чего подвергается отбору, скрещиванию и мутации. Такой метод ухудшает способность алгоритма искать решение с помощью отбора гиперплоскостей, но зато возрастает вероятность того, что одна из особей попадет в область глобального максимума и после оптимизации окажется решением задачи.

7.4 Island Models

Островная модель (*island model*) — модель параллельного генетического алгоритма. Разобьем популяцию на несколько подпопуляций. Каждая их них будет развиваться отдельно с помощью некого генетического алгоритма. Таким образом, можно сказать, что мы расселили особи по нескольким изолированным островам.

Изредка (например, каждые 5 поколений) происходит миграция – острова обмениваются несколькими хорошими особями.



Так как населённость островов невелика, то подпопуляции будут склонны к преждевременной сходимости. Поэтому важно правильно установить частоту миграции:

- чересчур частая миграция (или миграция слишком большого числа особей) приведет к смешению всех подпопуляций, и тогда островная модель будет несильно отличаться от обычного ГА
- если миграция будет слишком редкой, то она не сможет предотвратить преждевременного схождения подпопуляций

Генетические алгоритмы стохастичны, поэтому при разных его запусках популяция может сходиться к разным хорошим решениям. Островная модель позволяет запустить алгоритм

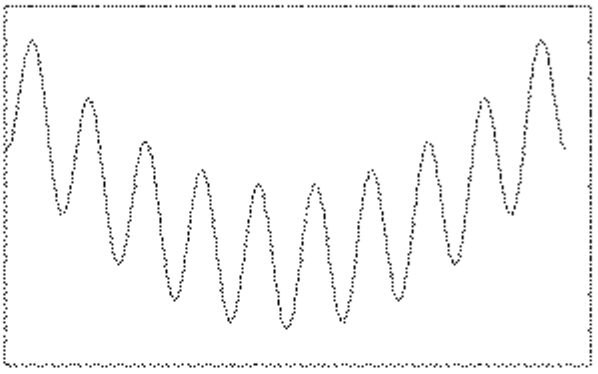
сразу несколько раз и совместить «достижения» разных островов для получения наилучшего решения.

8. Факторы, создающие сложность для ГА

Свойства функций приспособленности, создающие сложность для ГА.

- Многоэкстремальность: создается множество ложных аттракторов. Пример — функция Растригина:

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$$



На картинке изображен график функции Растригина с одним аргументом.

- *Обманчивость* (*deception*): функция построена так, что шаблоны малого порядка уводят популяцию к локальному экстремуму.

Пример: пусть строка состоит из 10-ти четырехбитных подстрок. Пусть u_i равно количеству единиц в i-той подстроке. Зададим функцию $g(u)$ следующей таблицей:

u	0	1	2	3	4
$g(u)$	3	2	1	0	4

и пусть функция приспособленности равна сумме $g(u_i)$ по всем $i = 1..10$:

$$f = \sum_{i=1}^{10} g(u_i)$$

- *Изолированность* («поиск иголки в стоге сена»): функция не предоставляет никакой информации, подсказывающей, в какой области искать максимум. Лишь случайное попадание особи в глобальный экстремум может решить задачу.

- *Дополнительный шум* (*noise*): значения приспособленности шаблонов сильно разбросаны, поэтому часто даже хорошие гиперплоскости малого порядка не проходят отбор, что замедляет поиск решения.

9. Выводы

- Генетические алгоритмы являются универсальным методом оптимизации многопараметрических функций, что позволяет решать широкий спектр задач.
- Генетические алгоритмы имеют множество модификаций и сильно зависят от параметров. Зачастую небольшое изменение одного из них может привести к неожиданному улучшению результата.
- Следует помнить, что применение ГА полезно лишь в тех случаях, когда для данной задачи нет подходящего специального алгоритма решения.

10. Ссылки

1. Авторский сайт Ю. Цоя (http://www.qai.narod.ru/).
2. Исаев С.А. Популярно о генетических алгоритмах (http://algolist.manual.ru/ai/ga/ga1.php).
3. http://www.gotai.net/ - сайт по ИИ.
4. http://neuronet.alo.ru/
5. http://www.neuroproject.ru/ – сайт компании, которая занимается разработкой программного обеспечения с использованием генетических алгоритмов и нейронных сетей.
6. Вороновский Г.К., Махотило К.В., Петрашев С.Н., Сергеев С.А., Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности, Харьков, ОСНОВА, 1997. – 112с.
7. Holland J. H. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence.— London: Bradford book edition, 1994 —211 p.
8. De Jong K.A. An analysis of the behavior of a class of genetic adaptive systems. Unpublished PhD thesis. University of Michigan, Ann Arbor, 1975. (Also University Microfilms No. 76-9381).
9. De Jong K.A., Spears W.M. An Analysis of the Interacting Roles of Population Size and Crossover // Proceedings of the International Workshop «Parallel Problems Solving from Nature» (PPSN'90), 1990.
10. De Jong K.A., Spears W.M. A formal analysis of the role of multi-point crossover in genetic algorithms. // Annals of Mathematics and Artificial Intelligence, no. 5(1), 1992.
11. Darrel Whitley "A Genetic Algorithm Tutorial", 1993.
12. Darrel Whitley, A Genetic Algorithm Tutorial, Statistics and Computing (4), 1994.
13. Darrel Whitley, An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls, Journal of Information and Software Technology, 2001.
14. Mitchell M. An Introduction to Genetic Algorithms. Cambridge, MA: The MIT Press, 1996.
15. K. Deb, S. Agrawal, Understanding Interactions Among Genetic Algorithm Parameters, 1998.
16. Robin Biesbroek "Genetic Algorithm Tutorial. 4.1 Mathematical foundations", 1999.
17. Soraya Rana "Examining the Role of Local Optima and Schema Processing in Genetic Search", 1999.
18. David E. Goldberg, Kumara Sastry "A Practical Schema Theorem for Genetic Algorithm Design and Tuning", 2001.
19. Koza, John R. Genetic programming: on the programming of computers by means of natural selection, A Bradford book, The MIT Press, London, 1992.