БИБЛИОТЕКИ ЯЗЫКА PYTHON ДЛЯ КОМПЬЮТЕРНЫХ ВЫЧИСЛЕНИЙ И МОДЕЛИРОВАНИЯ

Генетические алгоритмы и пакет DEAP в Python

Лобанов Алексей Владимирович
Главный специалист отдела разработки и внедрения АИС,
Ассистент кафедры информационных компьютерных технологий РХТУ им. Д.И. Менделеева

ПОЛЕЗНЫЕ РЕСУРСЫ

- Пакет DEAP (оф. сайт)
- Пакет DEAP (сайт)
- Пакет DEAP(github)

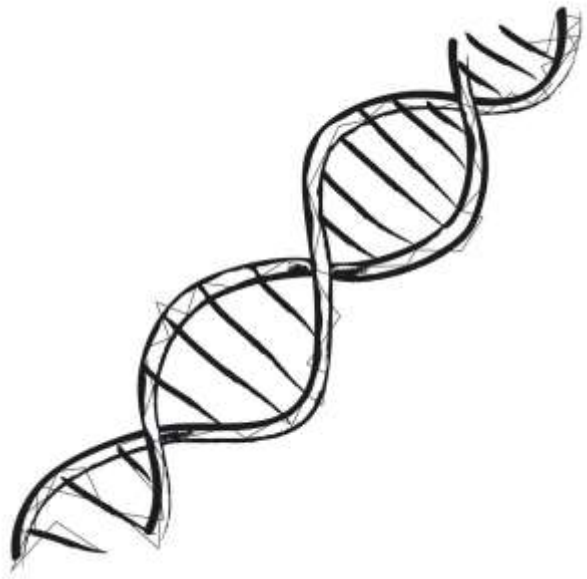
ТЕМЫ

- Основы генетических алгоритмов
- Пакет **DEAP**

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

Позаимствовавший идею у теории эволюции Чарльза Дарвина, один из самых удивительных методов решения задач заслуженно получил название «эволюционные вычисления». Самыми известными и широко распространенными представителями этого семейства являются генетические алгоритмы

Генетические алгоритмы – это семейство поисковых алгоритмов, идеи которых подсказаны принципами эволюции в природе. Имитируя процессы естественного отбора и воспроизводства, генетические алгоритмы могут находить высококачественные решения задач, включающих поиск, оптимизацию и обучение. В то же время аналогия с естественным отбором позволяет этим алгоритмам преодолевать некоторые препятствия, встающие на пути традиционных алгоритмов поиска и оптимизации, особенно в задачах с большим числом параметров и сложными математическими представлениями.

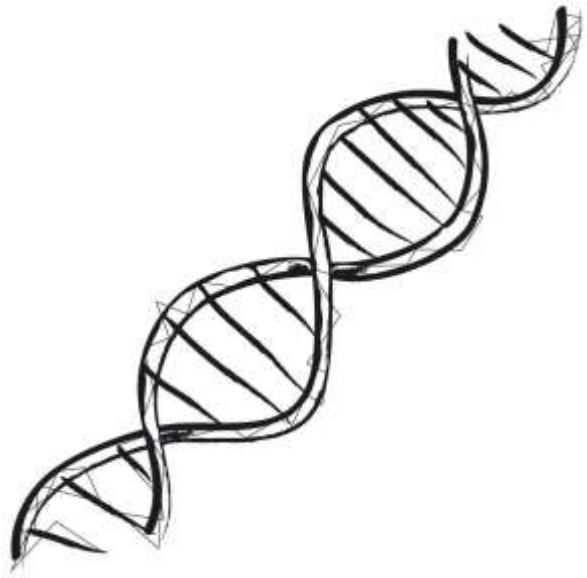


ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

Дарвиновская эволюция

Генетические алгоритмы реализуют упрощенный вариант дарвиновской эволюции. Ниже перечислены принципы эволюционной теории Дарвина.

- **Изменчивость.** Признаки (атрибуты) отдельных особей, входящих в состав популяции, могут изменяться. Поэтому особи отличаются друг от друга, например по внешнему виду или поведению.
- **Наследственность.** Некоторые свойства устойчиво передаются от особи к ее потомкам. Поэтому потомки похожи на своих родителей больше, чем на других особей, не связанных с ними родством.
- **Естественный отбор.** Обычно популяции борются за ресурсы, имеющиеся в окружающей их среде. Особи, обладающие свойствами, лучше приспособленными к окружающей среде, более успешны в борьбе за выживание и привносят больше потомков в следующее поколение.



Цель генетических алгоритмов – найти оптимальное решение некоторой задачи. Если дарвиновская эволюция развивает популяцию отдельных особей, то генетические алгоритмы развивают популяцию потенциальных решений данной задачи, называемых индивидуумами

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

Генотип

В природе скрещивание, воспроизводство и мутация реализуются посредством **генотипа** – набора генов, сгруппированных в хромосомы. Когда две особи скрещиваются и производят потомство, каждая хромосома потомка несет комбинацию генов родителей.

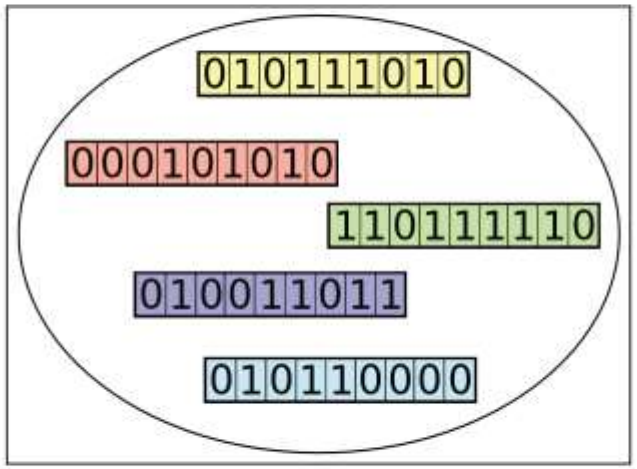
В случае генетических алгоритмов каждому индивидууму соответствует хромосома, представляющая набор генов. Например, хромосому можно представить двоичной строкой, в которой каждый бит соответствует одному **гену**:

Простое двоичное кодирование хромосомы

На рисунке выше показан пример такой двоично-кодированной хромосомы, представляющей одного индивидуума.

Популяция

В любой момент времени генетический алгоритм хранит популяцию **индивидуумов** – набор потенциальных решений поставленной задачи. Поскольку каждый индивидуум представлен некоторой хромосомой, эту популяцию можно рассматривать как коллекцию хромосом:



Популяция индивидуумов,
представленных двоично-кодированными хромосомами

Популяция всегда представляет текущее поколение и эволюционирует со временем, когда текущее поколение заменяется новым.

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

Функция приспособленности

На каждой итерации алгоритма индивидуумы оцениваются с помощью функции приспособленности (или целевой функции). Это функция, которую мы стремимся оптимизировать, или задача, которую пытаемся решить. Индивидуумы, для которых функция приспособленности дает наилучшую оценку, представляют лучшие решения и с большей вероятностью будут отобраны для воспроизводства и представлены в следующем поколении. Со временем качество решений повышается, значения функции приспособленности растут, а когда будет найдено удовлетворительное значение, процесс можно остановить

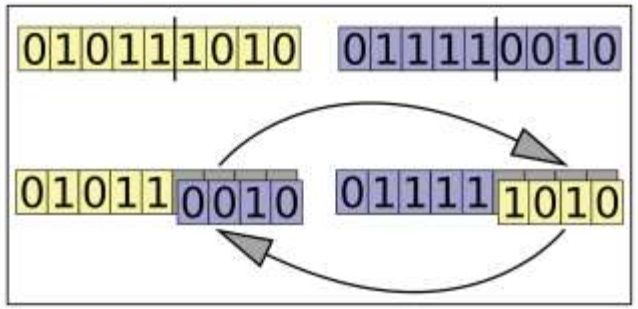
Отбор

После того как вычислены приспособленности всех индивидуумов в популяции, начинается процесс отбора, который определяет, какие индивидуумы будут оставлены для воспроизводства, т. е. создания потомков, образующих следующее поколение. Процесс отбора основан на оценке приспособленности индивидуумов. Те, чья оценка выше, имеют больше шансов передать свой генетический материал следующему поколению. Плохо приспособленные индивидуумы все равно могут быть отобраны, но с меньшей вероятностью. Таким образом, их генетический материал не полностью исключен.

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

Скрещивание

Для создания пары новых индивидуумов родители обычно выбираются из текущего поколения, а части их хромосом меняются местами (скрещиваются), в результате чего создаются две новые хромосомы, представляющие потомков. Эта операция называется скрещиванием, или рекомбинацией.



Операция скрещивания двух двоично-кодированных хромосом

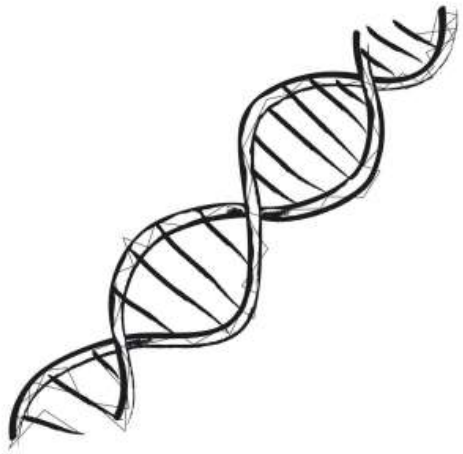
На рисунке выше показана операция скрещивания с созданием двух потомков родителей.

Мутация

Цель оператора мутации – периодически случайным образом обновлять популяцию, т. е. вносить новые сочетания генов в хромосомы, стимулируя тем самым поиск в неисследованных областях пространства решений. Мутация может проявляться как случайное изменение гена. Мутации реализуются с помощью внесения случайных изменений в значения хромосом, например инвертирования одного бита в двоичной строке.

Применение оператора мутации к двоично-кодированной хромосоме

На рисунке выше приведен пример операции мутации. Далее мы рассмотрим теорию, стоящую за генетическими алгоритмами.



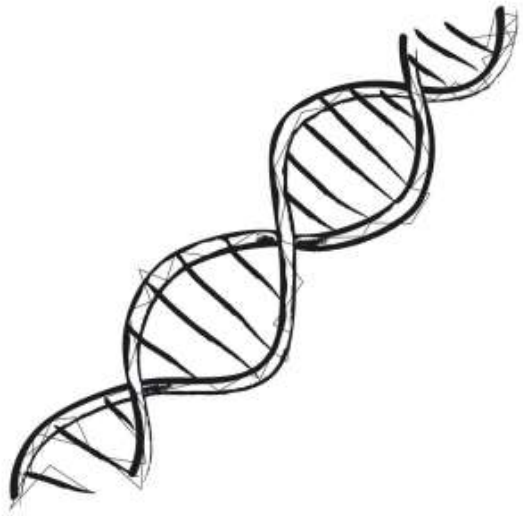
ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

Гипотеза структурных элементов, лежащая в основе генетических алгоритмов, заключается в том, что оптимальное решение задачи может быть собрано из небольших структурных элементов, и чем их больше, тем ближе мы подходим к оптимальному решению. Индивидуумам, которые содержат некоторые из желательных структурных элементов, назначается более высокая оценка. Повторные операции отбора и скрещивания приводят к появлению все лучших индивидуумов, передающих эти структурные элементы следующему поколению, возможно, в сочетании с другими успешными структурными элементами. Тем самым создается генетическое давление, направляющее популяцию в сторону появления все большего числа индивидуумов, обладающих структурными элементами, образующими оптимальное решение. В результате каждое поколение оказывается лучше предыдущего и содержит больше индивидуумов, близких к оптимальному решению.

ОТЛИЧИЯ ОТ ТРАДИЦИОННЫХ АЛГОРИТМОВ

Между генетическими и традиционными алгоритмами поиска и оптимизации имеется несколько важных различий. Перечислим основные характеристики генетических алгоритмов, отличающие их от традиционных:

- поддержание популяции решений:
- использование генетического представления решений;
- использование функции приспособленности;
- вероятностное поведение.



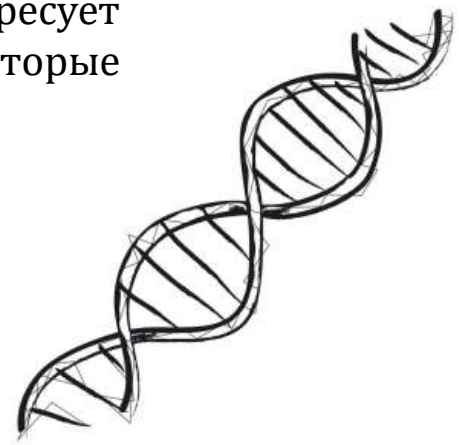
Целью генетического поиска является популяция потенциальных решений (индивидуумов), а не единственное решение. В любой точке поиска алгоритм сохраняет множество индивидуумов, образующих текущее поколение. На каждой итерации генетического алгоритма создается следующее поколение индивидуумов.

ОТЛИЧИЯ ОТ ТРАДИЦИОННЫХ АЛГОРИТМОВ

Генетические алгоритмы работают не с самими потенциальными решениями, а с их кодированными представлениями, которые часто называют **хромосомами**. Простым примером хромосомы является двоичная строка фиксированной длины. Хромосомы позволяют определить генетические операции скрещивания и мутации. Скрещивание реализуется обменом частей родительских хромосом, а мутация – изменением частей хромосом. Побочный эффект генетического представления – отделение поиска от исходной предметной области. Генетические алгоритмы не знают, что именно представляют хромосомы, и не пытаются их интерпретировать.

Функция приспособленности представляет **проблему**, которую мы пытаемся решить. Цель генетического алгоритма – найти индивидуумов, для которых оценка, вычисляемая функцией приспособленности, максимальна. В отличие от традиционных алгоритмов поиска, генетические алгоритмы анализируют только значение, возвращенное функцией приспособленности, их не интересует ни производная, ни какая-либо другая информация. Поэтому они могут работать с функциями, которые трудно или невозможно продифференцировать.

Многие традиционные алгоритмы по природе своей детерминированы, тогда как правила, применяемые генетическими алгоритмами для перехода от предыдущего поколения к следующему, вероятностные.
Несмотря на вероятностную природу процесса, поиск, основанный на генетическом алгоритме, нельзя назвать случайным; случайность используется, чтобы направить поиск в сторону тех областей пространства поиска, где выше шансы улучшить результаты. Теперь рассмотрим преимущества генетических алгоритмов.



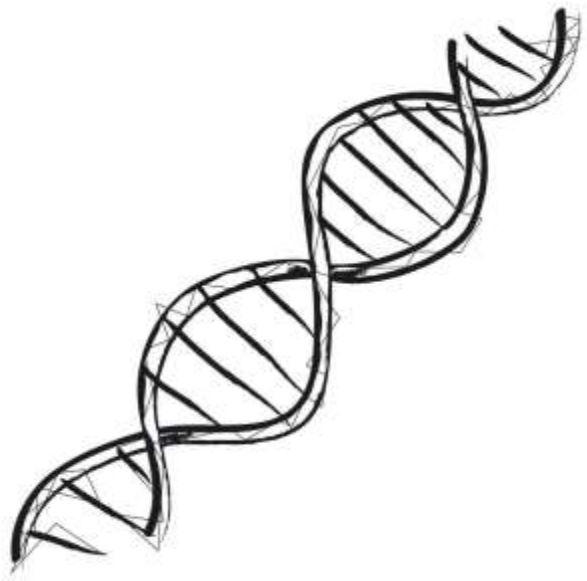
ГЕНЕТИЧЕСКИЕ АЛГОРИТМОВ

Преимущества генетических алгоритмов

- способность выполнять глобальную оптимизацию;
- применимость к задачам со сложным математическим представлением;
- применимость к задачам, не имеющим математического представления;
- устойчивость к шуму;
- поддержка распараллеливания и распределенной обработки;
- пригодность к непрерывному обучению.

Ограничения генетических алгоритмов

- необходимы специальные определения;
- необходима настройка гиперпараметров;
- большой объем счетных операций;
- опасность преждевременной сходимости;
- отсутствие гарантированного решения.



СЦЕНАРИИ ПРИМЕНЕНИЯ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

Генетические алгоритмы лучше применять для решения следующих задач:

- **Задачи со сложным математическим представлением**. Поскольку генетическим алгоритмам нужно знать только значение функции приспособленности, их можно использовать для решения задач, в которых целевую функцию трудно или невозможно продифференцировать, задач с большим количеством параметров и задач с параметрами разных типов.
- **Задачи, не имеющие математического представления**. Генетические алгоритмы не требуют математического представления задачи, коль скоро можно получить значение оценки или существует метод сравнения двух решений.
- **Задачи с зашумленной окружающей средой**. Генетические алгоритмы устойчивы к зашумленным данным, например прочитанным с датчика или основанным на оценках, сделанных человеком
- **Задачи, в которых окружающая среда изменяется во времени**. Генетические алгоритмы могут адаптироваться к медленным изменениям окружающей среды, поскольку постоянно создают новые поколения, приспосабливающиеся к изменениям.

С другой стороны, если для задачи известен специализированный способ решения традиционным или аналитическим методом, то вполне вероятно, что он окажется эффективнее.

БАЗОВАЯ СТРУКТУРА ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Начальная популяция состоит из случайным образом выбранных потенциальных решений (индивидуумов). Поскольку в генетических алгоритмах индивидуумы представлены хромосомами, начальная популяция – это, по сути дела, набор хромосом.

Для каждого индивидуума вычисляется функция приспособленности. Это делается один раз для начальной популяции, а затем для каждого нового поколения после применения операторов отбора, скрещивания и мутации. Поскольку приспособленность любого индивидуума не зависит от всех остальных, эти вычисления можно производить параллельно.

Применение *генетических операторов* к популяции приводит к созданию новой популяции, основанной на лучших индивидуумах из текущей.

Оператор отбора отвечает за отбор индивидуумов из текущей популяции таким образом, что предпочтение отдается лучшим.

Оператор скрещивания (или рекомбинации) создает потомка выбранных индивидуумов. Обычно для этого берутся два индивидуума, и части их хромосом меняются местами, в результате чего создаются две новые хромосомы, представляющие двух потомков.

Оператор мутации вносит случайные изменения в один или несколько генов хромосомы вновь созданного индивидуума. Обычно вероятность мутации очень мала.

УСЛОВИЯ ОСТАНОВКИ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Основные условия:

- Достигнуто максимальное количество поколений. Это условие заодно позволяет ограничить время работы алгоритма и потребление им ресурсов системы;
- На протяжении нескольких последних поколений не наблюдается заметных улучшений. Это можно реализовать путем запоминания наилучшей приспособленности, достигнутой в каждом поколении, и сравнения наилучшего текущего значения со значениями в нескольких предыдущих поколениях. Если разница меньше заранее заданного порога, то алгоритм можно останавливать.

Другие возможные условия:

- с момента начала прошло заранее определенное время;
- превышен некоторый лимит затрат, например процессорного времени или памяти;
- наилучшее решение заняло часть популяции, большую заранее заданного порога.

ПАКЕТ DEAP

Последние версии DEAP можно использовать с Python 2 и 3.
Каркас DEAP рекомендуется устанавливать с помощью программ easy_install или pip, например:
pip install deap

Для работы с генетическими алгоритмами создан целый ряд каркасов на Python, например GAFT, Pyevolve и PyGMO. Но каркас DEAP, в сравнении с состальными прост в использовании и предлагает широкий набор функций, поддерживает расширяемость и может похвастаться подробной документацией.

DEAP (сокращение от Distributed Evolutionary Algorithms in Python – распределенные эволюционные алгоритмы на Python) поддерживает быструю разработку решений с применением генетических алгоритмов и других методов эволюционных вычислений. DEAP предлагает различные структуры данных и инструменты, необходимые для реализации самых разных решений на основе генетических алгоритмов.
Каркас DEAP был разработан в канадском университете Лаваля в 2009 г. и предлагается на условиях лицензии GNU Lesser General Public License (LGPL).

ПАКЕТ DEAP

Класс creator

Метафабрика. позволяющая расширять существующие классы. Используется обычно для создания классов `Fitness` и `Individual`

```
from deap import creator, base, tools, algorithms

class Employee:
    pass

creator.create("Developer", Employee, position="Developer", programmingLanguages=set)

help(creator.Developer)
```

ПАКЕТ DEAP

Fitness

В этом классе инкапсулированы значения приспособленности. Приспособленность в deap можно определять по нескольким компонентам (целями). у каждой из которых есть вес. Комбинация весов определяет поведение или стратегию приспособления в конкретной задаче

base.Fitness - абстрактный класс. содержащий кортеж `weights`. Чтобы определить стратегию, кортежу наджо присвоить значения.

```
creator.create('FitnessMax', base.Fitness, weights=(1.0,))

help(creator.FitnessMax)
```

ПАКЕТ DEAP

В данном случае стратегия `FitnessMax` - максимизировать приспособленность индивидумов с единственной целью. Минимизация будет выглядеть так:

```
creator.create('FitnessMin', base.Fitness, weights=(-1.0,))
```

Множество целей. Первые две компоненты максимизируются, третья минимизируется. Важность компонент следует слева направо

```
creator.create('FitnessCompound', base.Fitness, weights=(1.0, 0.2, -0.5))
```

Кортеж values хранит сами значения приспособленности. Эти значения дает отдельно определяемая функция, которую обычно называют `evaluate()`. Кортеж содержит по одному значению для каждой функуции (цели).

Третий кортеж wvalues содержит взвешенные значени, полученные перемножением `values` и `weights`. Это используется для сравнения индивидуумов.

wvalues можно сравнивать лексикографически с помощью операторов `>`, `<`, `>=`, `<=`, `==`, `!=`

ПАКЕТ DEAP

Individual

С помощью этого класса определяются индивидуумы, образующие популяцию. В данном случае все гены индивидумов будут иметь тип лист, а класс каждого индивидума будет содержать экземляр `FitnessMax`

```
creator.create('Individual', list, fitness=creator.FitnessMax)

help(creator.Individual)
```

ПАКЕТ DEAP

Toolbox

Это контейнер для функций и операторов и позволяет создавать новые операторы путем назначения псевдонимов

Первым аргументом реигстрируем имя ф-ии. Вторым передаем ей выполняемую функцию. Остальные рагументы - необязательны (аргументы выполняемой ф-ии)

```
def sum_of_two(a, b):
    return a + b

toolbox = base.Toolbox()
toolbox.register('increment_by_five', sum_of_two, b=5)
toolbox.increment_by_five(10)

>>>15
```

ПАКЕТ DEAP

Создание генетических операторов

модуль `tools` содержит полезные функции для осуществления операций отбора, включая скрещивание и мутации, а так-же утилиты для инициализации. Поэтому `Toolbox` используется в основном для этого

В данном случае:

- создан оператор `tools.select` использующий турнирный отбор с аргументом 3 (размер тиурнира)
- создан оператор `mate` как псевдоним ф-ии `cxTwoPoint()`, выполняющей двухточечное скрещивание
- создан оператор `mutate` с операцией инвертирования бита и вероятностью 0.02

Функции отбора хранятся в `selection.py`. Функции скрещивания в `crossower.py`. Функции мутации в `mutation.py`

```
toolbox.register('select', tools.selTournament, tournize=3)
toolbox.register('mate', tools.cxESTwoPoint)
toolbox.register('mutate', tools.mutFlipBit, indpb=0.02)
```

ПАКЕТ DEAP

Создание популяции

`init.py` содержит несколько ф-ий полезных для создания и инициализации популяции.

Функцияя `initRepeat()` принимает три аргумента:

- тип контейнера результирующих объектов
- ф-ия. генерирующая объекты, которые помещаются в контейнер
- сколько объектов генерировать

```
import random
toolbox.register('zero_or_one', random.randint, 0, 1)
rnd = tools.initRepeat(list, toolbox.zero_or_one, 30)
rnd

>>> [0, 1, 0, 1, 1, ..., 1, 1]
```

ПАКЕТ DEAP

Вычисление приспособленности

значение приспособленности обычно возвращаются отдельно определенной ф-ией, которая регистрируется в `toolbox` как `evaluate` (соглашение)

```
def some_calc(nums):
    return nums

toolbox.register('evaluate', some_calc)
```

ПАКЕТ DEAP

Statistics

Класс `tools.Statistics` позволяет собират ьстатистику. задавая ф-ию, применяемую к данным, для которых вычисляется статистика. Например для случая, когда популяция является данными, функция извлекающая приспособленность каждого индивидуума и зарегистрированные методы

```
import numpy as np
stats = tools.Statistics(lambda ind: inf.fitness.values)
stats.register("max", np.max)
stats.register("avg", np.mean)
```

ПАКЕТ DEAP

Встроенные алгоритмы

В DEAP несколько встроенных эволюционных алгоритмов, реализующих пайплайн обучения. Пример примененеия:

```
population, logbook = algorithms.eaSimple(
    population,
    toolbox,
    cxpb=CONST1,
    mutpb=CONST2,
    ngen=CONST3,
    stats=stats,
    verbose=True,
)
```

ПАКЕТ DEAP

logbook

логбук - собранная статистика, которую возвращает алгоритм. Ее можно извлечь методом `select()` (в данном случае как раз и используются зарегистрирвоанные нами методы сбора статы)

```
max_, mean_ = logbook.select('max', 'avg`)
```

Зал славы

Класс HallOfFame модуля tools возвращает отранжированный список лучших индивидумов, даже если в процессе эволюции или отбора они были утрачены.

ЗАДАЧА ONEMAX

OneMax (или One-Max) – это простая задача оптимизации, которую часто приводят в пример как аналог программы «Hello World» в мире генетических алгоритмов. В этой главе мы будем использовать ее для демонстрации возможностей каркаса DEAP.

Задача OneMax состоит в том, чтобы найти двоичную строку заданной длины, для которой сумма составляющих ее цифр максимальна. Например, при решении задачи OneMax длины 5 будут рассматриваться такие кандидаты:

- 10010 (сумма цифр = 2);
- 01110 (сумма цифр = 3);
- 11111 (сумма цифр = 5).

Очевидно (нам), что решением всегда является строка, состоящая из одних единиц. Но генетический алгоритм не обладает таким знанием, поэтому должен слепо искать решение, пользуясь генетическими операторами. Если алгоритм справится с работой, то найдет решение (или приближение к нему) за разумное время.

КОМБИНАТОРНАЯ ОПТИМИЗАЦИЯ

Решение задачи коммивояжера

Допустим, вы управляете небольшим центром обработки и исполнения заказов и должны доставлять посылки заказчикам, располагая всего одним автомобилем. Как оптимально выбрать маршрут, чтобы объехать всех заказчиков и вернуться в исходную точку? Это пример классической задачи коммивояжера.

Задача коммивояжера восходит к 1930 году и является одной из наиболее изученных задач оптимизации. Зачастую она используется для оценки алгоритмов оптимизации. У этой задачи много вариантов, но первоначально она ставилась на примере коммивояжера, которому требуется объехать несколько городов:

Пусть дан список городов и известны расстояния между каждыми двумя городами. Найти кратчайший путь, проходящий через все города и возвращающийся в исходную точку.

Легко показать, что количество возможных путей, проходящих через n городов, равно $(n - 1)!/2$.

На рисунке ниже показан кратчайший путь, проходящий через 15 крупнейших городов Германии.

Кратчайший маршрут коммивояжера,
проходящий через 15 крупнейших городов Германии

Источник: https://commons.wikimedia.org/wiki/File:TSP_Deutschland_3.png.
Автор Kapitan Nemo. Является общественным достоянием

Поскольку в этом случае $n = 15$, количество возможных путей равно $14!/2 = 43\,589\,145\,600$ – пугающее число.

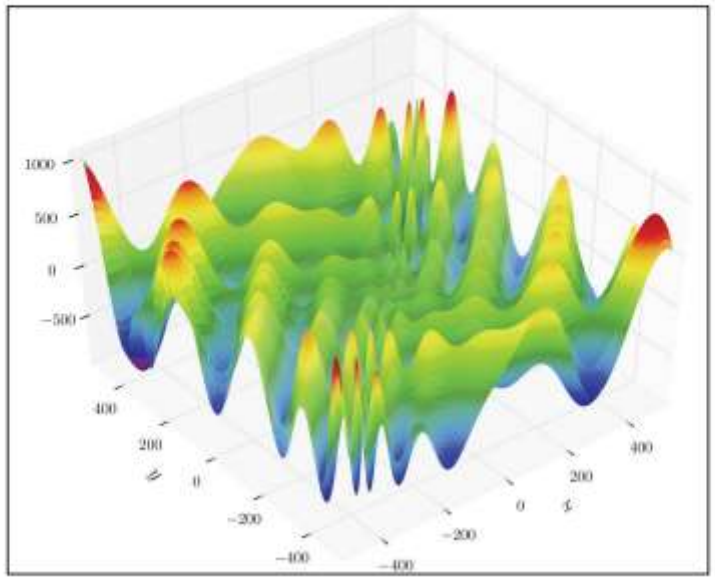
В контексте алгоритмов поиска каждый путь (или частичный путь), проходящий через города, представляет *состояние*, а множество всех возможных путей рассматривается как *пространство состояний*. У каждого пути имеется стоимость – его длина, – и мы ищем путь минимальной стоимости.

Как уже отмечалось, пространство состояний очень велико даже при сравнительно небольшом количестве городов, поэтому рассмотреть каждый путь не представляется возможным. И хотя проложить какой-то путь через все города сравнительно просто, найти оптимальный путь очень трудно.

ОПТИМИЗАЦИЯ НЕПРЕРЫВНЫХ ФУНКЦИЙ

Оптимизация функции Eggholder

Функция Eggholder (она называется так потому, что ее форма напоминает подставку для яиц) часто используется для тестирования алгоритмов оптимизации. Нахождение единственного глобального минимума этой функции считается трудной задачей из-за большого количества локальных минимумов.



Функция Eggholder

Математически функция описывается выражением

$$f(x,y) = -(y+47)\cdot\sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\cdot\sin\sqrt{\left|x-(y+47)\right|}$$

и обычно вычисляется в области поиска, ограниченной интервалом [−512, 512] по каждому измерению.

Известно, что глобального минимума эта функция достигает в точке

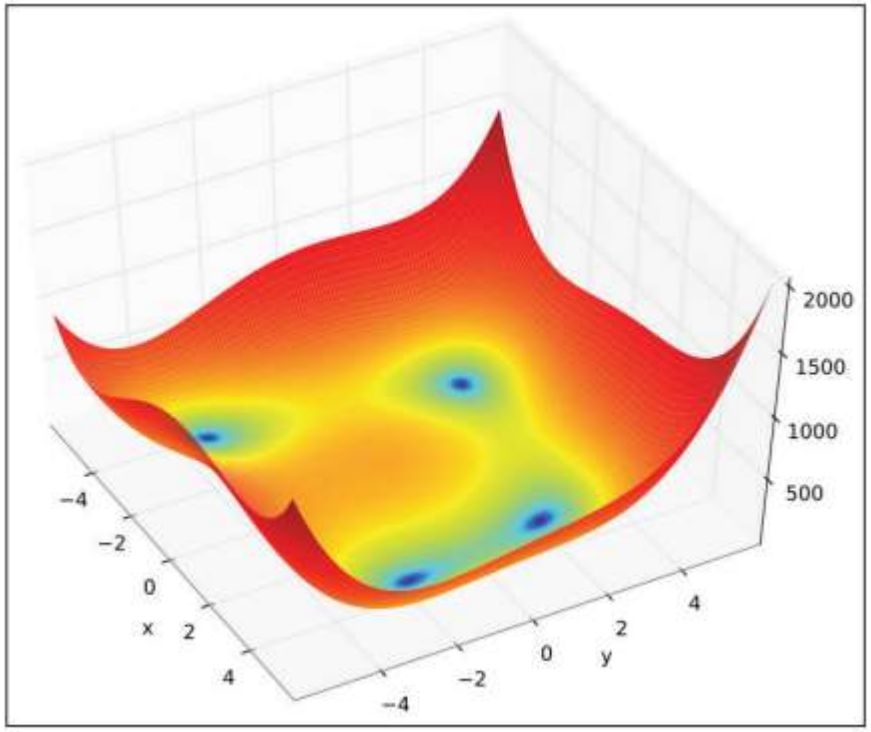
$x=512, y=404.2319,$

в которой равна −959.6407.

ОПТИМИЗАЦИЯ НЕПРЕРЫВНЫХ ФУНКЦИЙ

Оптимизация функции Химмельблау

На рисунке ниже изображена функция Химмельблау, которая также часто используется для тестирования алгоритмов оптимизации.



Функция Химмельблау

Математически эта функция описывается следующей формулой:

$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$.

Обычно она вычисляется в области, ограниченной интервалом [−5, 5] по каждому измерению.

Хотя эта функция кажется проще функции Eggholder, особый интерес представляет ее *многомодальность*, т. е. наличие нескольких глобальных минимумов. Точнее, у функции имеется четыре глобальных минимума, равных 0, в следующих точках:

- $x = 3.0, y = 2.0$;
- $x = -2.805118, y = 3.131312$;
- $x = -3.779310, y = -3.283186$;
- $x = 3.584458, y = -1.848126$.

ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

Реализовать оптимизацию одной из функций на выбор:

- Функция Гольдшейна-Прайса
- Функция Бута
- Функция Матьяса
- Функция Химмельблау
- Функция Стыбинского-Танга

Найти глобальный минимум этой функции.

СПАСИБО ЗА ВНИМАНИЕ!