# Практическая работа №7.Генетический алгоритм

***Что такое генетические алгоритмы?***

Генетические алгоритмы (GA) — это алгоритмы поиска, основанные на понятиях естественного отбора и генетики. GA — это подмножество гораздо большей ветви вычислений, известной как эволюционные вычисления.

GA были разработаны Джоном Холландом и его студентами и коллегами из Мичиганского университета, прежде всего Дэвидом Э. Голдбергом. С тех пор он был опробован на различных задачах оптимизации с высокой степенью успеха.

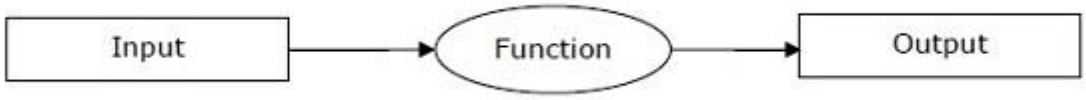
В ГА у нас есть пул возможных решений данной проблемы. Эти решения затем подвергаются рекомбинации и мутации (как в естественной генетике), рождают новых детей, и процесс повторяется для разных поколений. Каждому индивидууму (или подходящему решению) присваивается значение пригодности (в зависимости от значения его целевой функции), и более подходящие индивидуумы получают более высокий шанс для спаривания и получения более **подходящих** индивидуумов. Это соответствует дарвиновской теории **выживания наиболее приспособленных** .

Таким образом, он продолжает **эволюционировать от** лучших людей или решений на протяжении поколений, пока не достигнет критерия остановки.

Генетические алгоритмы по своей природе достаточно рандомизированы, но они работают намного лучше, чем случайный локальный поиск (где мы просто пробуем случайные решения, отслеживая лучшие на данный момент), поскольку они также используют историческую информацию.

***Как использовать GA для задач оптимизации?***

Оптимизация — это действие, позволяющее сделать дизайн, ситуацию, ресурс и систему максимально эффективными. Следующая блок-схема показывает процесс оптимизации —




***Этапы механизма GA для процесса оптимизации***

Ниже приведена последовательность шагов механизма GA при использовании для оптимизации задач.

- Шаг 1 — Генерация начальной популяции случайным образом.
- Шаг 2 — Выберите исходное решение с наилучшими значениями пригодности.
- Шаг 3 — рекомбинируйте выбранные решения, используя операторы мутации и кроссовера.
- Шаг 4 — Вставьте потомство в популяцию.
- Шаг 5 — Теперь, если условие остановки выполнено, верните решение с наилучшим значением пригодности. Остальное перейдите к шагу 2.

***Установка необходимых пакетов***

Для решения проблемы с использованием генетических алгоритмов в Python мы будем использовать мощный пакет для GA, называемый **DEAP** . Это библиотека новой эволюционной вычислительной среды для быстрого прототипирования и проверки идей. Мы можем установить этот пакет с помощью следующей команды в командной строке

```
pip install deap
```

Если вы используете среду **anaconda** , то для установки deap можно использовать следующую команду:

```
conda install -c conda-forge deap
```

**Реализация решений с использованием генетических алгоритмов**

***Генерация битовых паттернов***

В следующем примере показано, как сгенерировать битовую строку, которая будет содержать 15 строк, на основе проблемы **One Max** .Импортируйте необходимые пакеты, как показано ниже

```
importrandom
fromdeapimportbase, creator, tools
```

Определите функцию оценки. Это первый шаг к созданию генетического алгоритма.

```
def eval_func(individual):
   target_sum = 15
   return len(individual) - abs(sum(individual) - target_sum),
```

Теперь создайте набор инструментов с правильными параметрами —

```
def create_toolbox(num_bits):
   creator.create("FitnessMax", base.Fitness, weights=(1.0,))
   creator.create("Individual", list, fitness=creator.FitnessMax)
```

Инициализировать панель инструментов

```
   toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual,
   toolbox.attr_bool, num_bits)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

Операторoценки

```
toolbox.register("evaluate", eval_func)
```

Операторкроссовера

```
toolbox.register("mate", tools.cxTwoPoint)
```

Оператормутации

```
toolbox.register("mutate", tools.mutFlipBit, indpb = 0.05)
```

Оператордляразведения —

```
toolbox.register("select", tools.selTournament, tournsize = 3)
return toolbox
if __name__ == "__main__":
   num_bits = 45
   toolbox = create_toolbox(num_bits)
   random.seed(7)
   population = toolbox.population(n = 500)
   probab_crossing, probab_mutating = 0.5, 0.2
   num_generations = 10
   print('\nEvolution process starts')
```

Оценкавсегонаселения

```
fitnesses = list(map(toolbox.evaluate, population))
for ind, fit in zip(population, fitnesses):
   ind.fitness.values = fit
print('\nEvaluated', len(population), 'individuals')
```

Создадим и выведем на печать поколения

```
for g in range(num_generations):
   print("\n- Generation", g)
```

Выбор следующего поколения

```
offspring = toolbox.select(population, len(population))
```

Теперь, клонируем

```
offspring = list(map(toolbox.clone, offspring))
```

Применим кроссовер и мутацию на потомство —

```
for child1, child2 in zip(offspring[::2], offspring[1::2]):
   if random.random() < probab_crossing:
toolbox.mate(child1, child2)
```

Удалить значение фитнеса ребенка

```
del child1.fitness.values

del child2.fitness.values
```

Теперь применим мутацию

```
for mutant in offspring:
   if random.random() < probab_mutating:
   toolbox.mutate(mutant)
   del mutant.fitness.values
```

Оценимособи с недопустимой пригодностью

```
invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
fitnesses = map(toolbox.evaluate, invalid_ind)
for ind, fit in zip(invalid_ind, fitnesses):
   ind.fitness.values = fit
print('Evaluated', len(invalid_ind), 'individuals')
```

Теперь заменим население на следующее поколение

```
population[:] = offspring
```

Распечатать статистику по текущим поколениям

```
fits = [ind.fitness.values[0] for ind in population]
length = len(population)
mean = sum(fits) / length
sum2 = sum(x*x for x in fits)
std = abs(sum2 / length - mean**2)**0.5
print('Min =', min(fits), ', Max =', max(fits))
print('Average =', round(mean, 2), ', Standard deviation =',
round(std, 2))
print("\n- Evolution ends")
```

Окончательный вывод

```
   best_ind = tools.selBest(population, 1)[0]
   print('\nBest individual:\n', best_ind)
   print('\nNumber of ones:', sum(best_ind))
Following would be the output:
Evolution process starts
Evaluated 500 individuals
- Generation 0
Evaluated 295 individuals
Min = 32.0 , Max = 45.0
Average = 40.29 , Standard deviation = 2.61
- Generation 1
Evaluated 292 individuals
Min = 34.0 , Max = 45.0
Average = 42.35 , Standard deviation = 1.91
- Generation 2
Evaluated 277 individuals
Min = 37.0 , Max = 45.0
Average = 43.39 , Standard deviation = 1.46
… … … …
- Generation 9
Evaluated 299 individuals
Min = 40.0 , Max = 45.0
Average = 44.12 , Standard deviation = 1.11
- Evolution ends
Best individual:
[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1]
Number of ones: 15
```