|  |
| --- |
|  |
|  |
| |  | | --- | |  | | МИНОБРНАУКИ РОССИИ | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет****"**  **РТУ МИРЭА** | |  | |  | |
| Институт информационных технологий (ИТ) |
| Кафедра инструментального и прикладного программного обеспечения (ИиППО) |

|  |
| --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 3** |
| **по дисциплине** |
| **«Программное обеспечение интеллектуальных систем**»  *Генетический алгоритм* |

|  |  |
| --- | --- |
|  |  |
| Выполнил студент группы ИКБО-02-17 | *Крамаренко З.В* |
| Приняла | *Зорина Н.В.* |

|  |  |  |
| --- | --- | --- |
| Работа выполнена | «\_\_\_» \_\_\_\_\_\_\_\_2020 |  |
|  |  |  |
| «Зачтено» | «\_\_\_» \_\_\_\_\_\_\_\_2020 |  |

**Цель:** Ознакомиться с работой генетических алгоритмов на примере

создания приложений.

**Задание:**

В рамках лабораторной работы попробуйте решить одну из следующих

задач:

1. Тренировка нейросети;

2. Решение NP-полной задачи.

**Ход работы**

Данный генетическим алгоритм должен из начального набора символов, сформированных случайным образом, получить строку "Genetic Algorithm!" .

Что он делает? Он берет случайные строки мутирует и скрещивает их до тех пор, пока не получится нужная строка. Т.е. генетический алгоритм работает наподобие эволюции. Сначала создаются начальные популяции, затем они скрещиваются между собой (при этом возможно возникновение мутаций). Популяции, выжившие в процессе естественного отбора, проверяются на удовлетворение заданным критериям. Если удовлетворяют — то конец, если нет — вновь скрещиваются и так до финальной победы.

***Формализуем задачу со случайной строкой***

Входные данные: строка S

Выходные данные: натуральное число N, равное количеству поколений необходимых для преобразования случайной строки длины len(S) в строку S

Под мутацией строки S мы понимаем замену одного случайно выбранного символа из строки S на другой произвольный символ алфавита. В данной задаче мы используем только символы нижнего регистра латиницы и пробел.

Изначальная популяция - случайная строка равная по длине входной строке S.

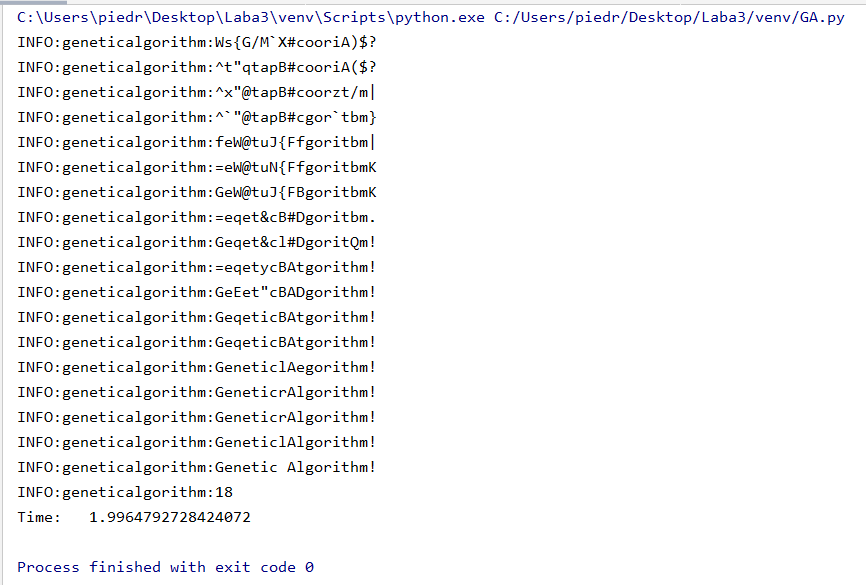
Потомки. Пусть мы зафиксировали количество мутаций одной строки на константу k1, тогда потомки — это k1 мутаций каждой строки текущего поколения.

Выжившие. Пусть мы зафиксировали размер популяции на константу k2, тогда выжившие — это k2 строк максимально похожих на входную строку S.

Исходный код

**import** sys  
**import** time  
**import** string  
**import** random  
**import** logging  
  
**from** optparse **import** OptionParser  
logger = logging.getLogger(**'geneticalgorithm'**)  
  
GENES = **""**.join(map(**lambda** x, y: x + y, string.ascii\_uppercase, string.ascii\_lowercase)) + \  
 string.punctuation + **" "**GOAL = **"Genetic Algorithm!"  
  
def** fitness(dnk, goal):  
 f = 0  
 **for** index, gene **in** enumerate(dnk):  
 **if** gene != goal[index]:  
 f -= 1  
 **return** f  
  
**def** sample\_wr(population, k):  
 n = len(population)  
 \_random, \_int = random.random, int  
 result = [**None**] \* k  
 **for** i **in** range(k):  
 j = \_int(\_random() \* n)  
 result[i] = population[j]  
 **return** result  
  
  
**class** GeneticCode:  
 **def** \_\_init\_\_(self, dnk=**""**, goal=GOAL):  
 **if** dnk == **""**:  
 self.dnk = **""**.join(sample\_wr(GENES, len(goal)))  
 **else**:  
 self.dnk = dnk  
 self.goal = goal  
  
 **def** get(self):  
 **return** self.dnk  
  
 **def** fitness(self):  
 **return** fitness(self.dnk, self.goal)  
  
 **def** mutate(self, turns=5):  
 \_dnk = list(self.dnk)  
 **for** item **in** range(turns):  
 rnd\_elem\_index = random.randint(0, len(\_dnk) - 1)  
 **if** \_dnk[rnd\_elem\_index] == self.goal[rnd\_elem\_index]:  
 **pass  
 else**:  
 \_dnk[rnd\_elem\_index] = random.choice(GENES)  
 self.dnk = **""**.join(\_dnk)  
  
 **def** replicate(self, another\_dnk):  
 part = random.randint(0, len(self.dnk) - 1)  
 **return ""**.join(self.dnk[0:part] + another\_dnk.get()[part:])  
  
**class** GenePool():  
 pool\_size = 100  
  
 **def** \_\_init\_\_(self, goal=GOAL):  
 self.pool = [GeneticCode(goal=goal) **for** item **in** range(self.pool\_size)]  
 self.goal = goal  
  
 **def** \_print(self):  
 **for** item **in** self.pool:  
 print  
 item.get() + **" - "** + str(item.fitness())  
  
 **def** get\_random(self):  
 **return** self.pool[random.randint(0, len(self.pool) - 1)]  
  
 **def** darvin(self, winners=0.1):  
 all\_fitness = [(item.fitness(), item) **for** item **in** self.pool]  
 new\_pool = [item[1] **for** item **in** sorted(all\_fitness, key=**lambda** x: x[0], reverse=**True**)]  
 self.pool = new\_pool[:int(round(self.pool\_size \* winners))]  
  
 **while** len(self.pool) < self.pool\_size:  
 new\_life = self.get\_random().replicate(self.get\_random())  
 new\_gc = GeneticCode(dnk=new\_life, goal=self.goal)  
 self.pool.append(new\_gc)  
  
 **def** evolution(self, turns=1000):  
 iterations = 0  
 **while** (iterations < turns) **and** (self.pool[0].get() != self.goal):  
 **for** index, item **in** enumerate(self.pool):  
 self.pool[index].mutate()  
 self.darvin()  
 logger.info(self.pool[0].get())  
 time.sleep(0.1)  
 iterations += 1  
  
 **return** iterations  
  
  
**def** main():  
 usage = **'%s [options] [text]'** % sys.argv[0]  
 parser = OptionParser(usage)  
 parser.add\_option(**'-l'**, **'--log'**, default=**'-'**,  
 help=**'redirect logs to file'**)  
 opts, args = parser.parse\_args()  
  
 **if** opts.log == **'-'**:  
 logging.basicConfig(stream=sys.stdout, level=logging.INFO)  
 **else**:  
 logging.basicConfig(filename=**"geneticalgorithm.log"**, level=logging.INFO)  
  
 **if** args:  
 text = args[0]  
 **else**:  
 text = GOAL  
  
 gp = GenePool(goal=text)  
 steps = gp.evolution()  
 logger.info(steps)  
  
  
start\_time = time.time()  
main()  
  
print(**"Time:\t%s"** % (time.time() - start\_time))

***Результаты работы программы***



Видно, что за 18 итераций нам удалось добиться заданной строки.

Ссылка на GitHub: <https://github.com/ZlataKr/POIS/blob/master/GA.py>

**Вывод**

В ходе выполнения лабораторной работы мы ознакомились с работой генетического алгоритма, а также применили свои знания на практике.

**Список литературы**

1. Себастьян Рашка. Python и машинное обучение / - М.: Изд-во Вильямс, 2019

2. Петер Флах. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / - М.: Изд-во ДМК Пресс, 2019

3. Вирсански Эйял. Генетический алгоритм на Python / - М.: Изд-во ДМК Пресс, 2020

4. Леонид Гладков. Генетические алгоритмы / - М.: Отдельное издание 2010

5. Материалы лекций