Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине «Интеллектуальные системы»

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

Бондарев Матвей Владимирович
Копылов Данила Олегович

Направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии
Направленность (профиль) «Искусственный интеллект и разработка программных продуктов»

Руководитель работы
_____А.Д. Брагин
подпись
«_____» _____ 20 ___ г.

Авторы работы
студент группы № 932204
_____ М.В. Бондарев
подпись
«_____» _____ 20 ___ г.

студент группы № 932101
_____ Д.О. Копылов
подпись
«_____» _____ 20 ___ г.

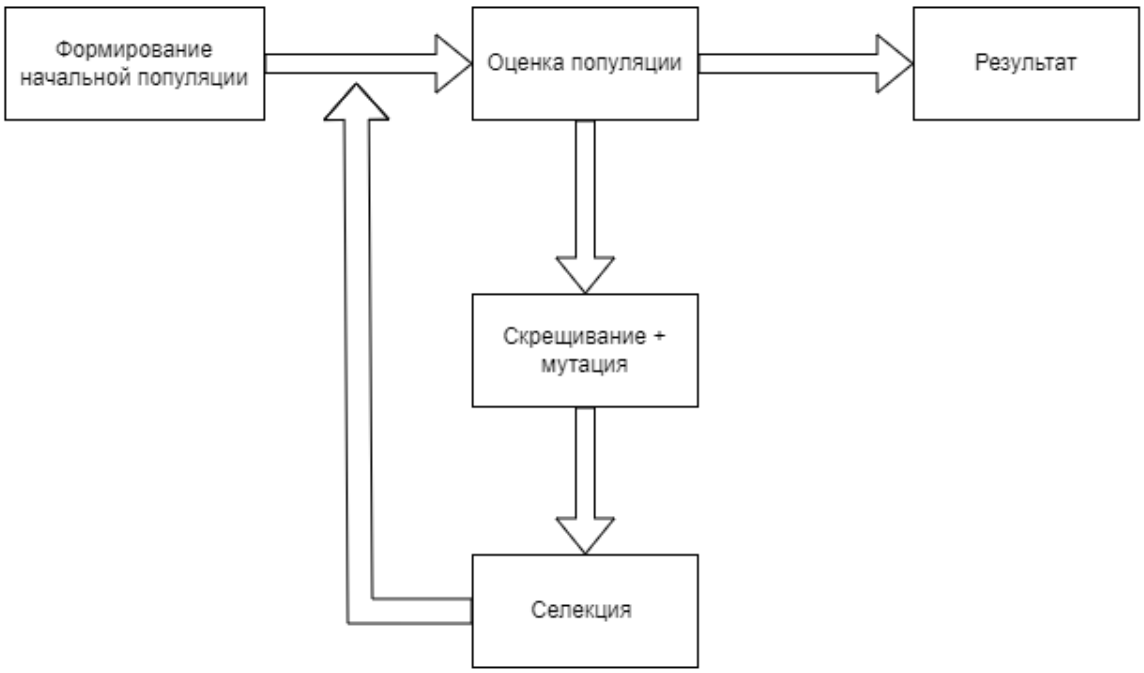
Томск – 2024

Цель и постановка задачи

Требуется создать программу, реализующую генетический алгоритм и найти с его помощью особь, гены которой соответствуют, в формате RGB, фиолетовому цвету [96, 96, 159]

Структурная схема алгоритма

Ниже приведена структурная схема алгоритма



В начале работы алгоритма популяция формируется случайным образом (блок «Формирование начальной популяции»).
По результатам оценивания особей наиболее приспособленные из них выбираются (блок «Селекция») для скрещивания.
В результате скрещивания(блок "Скрещивание + мутация")выбранных особей посредством применения генетического оператора кроссинговера создается потомство, генетическая информация которого формируется в результате обмена хромосомной информацией между родительскими особями, также в этом блоке происходит мутация особи. Созданные потомки формируют новую популяцию

Реализация генетического алгоритма

Для реализации алгоритма потребуется встроенный модуль random, а также itertools.

Для начала зададим класс Bacteria, в котором и будут происходить основные операции: создание объекта, мутация, скрещивание, также внутри класса будет вычисляться приспособленность особи.

Ниже приведен пример конструктора класса и его член данных:

```
class Bacteria:

    def __init__(self, R=True, G=True, B=True, target=[96, 96, 159]):

        if not all([R, G, B]):
            self.__R = r.randint(0, 256)
            self.__G = r.randint(0, 256)
            self.__B = r.randint(0, 256)

        else:
            self.__R = R
            self.__G = G
            self.__B = B

        self.__RT = target[0]
        self.__GT = target[1]
        self.__BT = target[2]
        self.__pm = 0.7
```

Конструктор срабатывает, если у особи не переданы параметры R, G, B
Также есть поля RT, GT, BT, в которых хранится информация о target особи, к которой популяция стремится.

Перегрузка оператора сложения выполняет роль скрещивания и мутации.

```
def __add__(self, other):
    gen = r.randint(0, 2)
    g_1 = self(); g_2 = other()
    gen_1 = g_1.pop(gen); gen_2 = g_2.pop(gen)
    new_gen = int("{0:b}".format(gen_1).zfill(4)[:4] + "
{0:b}".format(gen_2).zfill(8)[4:8], 2)
    combinations = [list(comb) for comb in itertools.product(g_1, g_2)]

    mutated_bact = []
    for comb in combinations:
        comb_c = comb.copy()
        if self.__pm > r.random():
            mutated_index = r.randint(0, 1)
            mutated = "{0:b}".format(comb[mutated_index]).zfill(8)
            bit_indices = r.sample(range(len(mutated)),r.randint(4, 8))
            inverted_num = ''.join([str((int(bit, 2) + 1) % 2) if i in
bit_indices else bit for i, bit in enumerate(mutated)])
            comb_c[mutated_index] = int(inverted_num, 2)
        mutated_bact.append(comb_c)

    final_out = []
    for comb in mutated_bact:
        comb_c = comb.copy()
        comb_c.insert(gen, new_gen)
        final_out.append(comb_c)

    return [Bacteria(*i, target=[self.__RT, self.__GT, self.__BT]) for
i in final_out]
```

Изначально выбирается один из трех генов двух бактерий, приводится к бинарному виду, после чего с помощью одноточечного кроссовера разделяется на два, где первую половину занимает часть гена первой особи, оставшуюся - второй.
Также с помощью itertools.product собираются комбинации и двух оставшихся генов каждой особи, случайно выбирается один из генов, после чего в цикле они приводятся к бинарному виду. Из бинарного представления выбирается от 4 до 8 цифр, после чего их представление инвертируется.

Полученные мутации записываются в новый список генов, после чего функция возвращает 4 новые особи.

Далее будет приведен код функции приспособленности особи: evaluete, которая вычисляется как Евклидово расстояние между двумя точками:

$$evaluete = \sqrt{(RT - R)^2 + (GT - G)^2 + (BT - B)^2}.$$

А также дополнительные перегрузки, которые потребуются в дальнейшей программе.

```
def _evaluete(self):
    return ((self.__RT - self.__R) ** 2 +
    (self.__GT - self.__G) ** 2 +
    (self.__BT - self.__B) ** 2) ** .5

def __gt__(self, other):
    return self._evaluete() < other._evaluete()

def __call__(self):
    return [self.__R, self.__G, self.__B]
```

Функция __gt__ отвечает за перегрузку оператора сравнения: >, сравнение происходит за счет функции evaluete.

Функция __call__ отвечает за перегрузку оператора вызова: (), требуется для отображения член данных класса.

На этом код класса Bacteria закончен, далее будет приведена функция, которая отвечает за селекцию. В дальнейшем мы вернемся к этой теме, когда будем рассматривать блок, где происходит поэтапный вызов функция для работы генетического алгоритма.

```
def tournament_selection(population):
    selected_parents = []
    for _ in range(len(population) // 3):
        top = min(r.sample(population, 4), key=Bacteria._evaluete)
        selected_parents.append(top)
    return selected_parents
```

В ходе турнирной селекции случайно происходит выбор четырех особей среди всей популяции, наиболее приспособленная особь добавляется в список selected_parents.

Далее приведем блок, где и происходит выполнение генетического алгоритма.

```
len_start_prop = 80

start_pop = [Bacteria(target=[96, 96, 159]) for i in
range(len_start_prop)]
selected = tournament_selection(start_pop)

counter = 0
while True:
    counter += 1
    for prop in selected:
        if prop._evaluete() == 0:
            print(f"Epoch = {counter}, Finded = {prop()}")
            break
    else:
        new_pop = []
        for i in range(len(selected)):
            if 0.6 > r.random():
                new_pop.extend(selected[i-1] + selected[i])
            else:
                new_pop.append(selected[i-1])
                new_pop.append(selected[i])
        selected = tournament_selection(new_pop)
        if counter % 20 == 0 and counter != 0:
            print(f"Epoch = {counter}", f"\nTop Fitness:
{top_fitness(selected)}")
            selected = sorted(selected, key=Bacteria._evaluete)
[:len_start_prop]
        continue
    break
```

Задаем начальную популяцию, в нашем случае размерность: 80, после чего происходит селекция, в цикле происходит скрещивание особей, в зависимости от выпадения числа r.random(), если скрещивание не происходит, исходные особи просто попадают в новую популяцию.

Далее проводится поиск target особи, если алгоритм ее не находит, происходит вызов турнирной селекции, также каждые 20 поколений мы урезаем популяцию до начальной размерности, по признаку приспособленности, так как популяция очень быстро растет.

Ниже приведен пример выполнения программы:

```
Epoch = 20 Top Fitness: [7.141, 10.05, 10.05]
Epoch = 40 Top Fitness: [2.0, 2.0, 2.0]
Epoch = 46, Finded = [96, 96, 159]
```

Как видно из примера выполнения программы, особь, соответствующая target, была найдена за 46 эпох.

Вывод

В ходе выполнения лабораторной работы были рассмотрены возможные схемы реализации генетического алгоритма, реализован сам генетический алгоритм, который реализует поиск особи, гены которой соответствуют,
в формате RGB, фиолетовому цвету [96, 96, 159]