МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Основы машинного обучения»

Тема: Генетические алгоритмы.

Вариант 2

Студент гр. 1303 Самохин К. А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Изучить работу генетических алгоритмов. Подобрать такие параметры, при которых решение находится почти для любых данных.

Постановка задачи.

Задача о распределении нагрузки станков. Имеется N (задается пользователем) станков на заводе по написанию красивого кода. Необходимо составить план работы для N станков на неделю, в какие дни и смены станок работает, а в какие нет. Неделя состоит из 7 дней, и каждый день по 2 смены (дневная и ночная). Известно, что станок не может работать больше 3 смен подряд, иначе код перестанет быть красивым, а также каждый день должно работать не менее K станков ($K_1, K_2, ..., K_7$ - задается пользователем). Также известно, что станки своенравные и не хотят работать в определенные дни (задается пользователем), но при необходимости можно заставить работать.

Задание.

1. Постановка задачи
   - 1.1. Формализовать описание задачи в математическом виде.
   - 1.2. Описать представление решения в виде хромосомы. Пояснить выбранное представление и какие ограничения есть на хромосому.
   - 1.3. Разработать метрику для оценки качества решения. Пояснить разработанную метрику, и как она позволяет соблюдать мягкие и жесткие ограничения.
2. Выполнение ГА
   - 2.1. Выбрать операции для ГА: метод отбора, метод скрещивания, метод мутации. Обосновать выбор.
   - 2.2. Подобрать параметры ГА: размер популяции, вероятность скрещивания и мутации, количество поколений (если есть другие параметры, то и их тоже). Описать процесс подбора параметров.

2.3. Провести оптимизацию и визуализировать полученное решение. *Графически отобразить решение. Например, траектория броска мяча или в виде таблицы показать расписание работы.*

2.4. Построить график зависимости лучше и средней приспособленности в зависимости от поколения.

3. Анализ ГА

3.1. Продемонстрировать случаи, когда находится точное решение без нарушения ограничений. Визуализируя как в п. 2.3.

3.2. Продемонстрировать случаи, когда находится точное решение с нарушением мягких ограничений. Визуализируя как в п. 2.3.

3.3. Продемонстрировать случаи, когда решение невозможно найти. Визуализируя, как в п. 2.3.

Выполнение работы.

1. Постановка задачи.

1.1. Формализуем описание задачи в математическом виде. При решении задачи оптимизации будут использованы следующие переменные:

- N – число имеющихся станков.
- D – количество дней (в неделе 7 дней, это значение и будем использовать).
- S – количество смен в сутках (2: дневная и ночная).
- K_i – минимальное количество работающих станков в i-й день.
- UNWANTED_DAYS – набор нежелательных дней для каждого станка.

В процессе анализа поставленной задачи были выявлены следующие ограничения:

- Станок не может работать более трёх смен подряд, далее код перестаёт быть красивым (считаем, что некрасивый код неприемлем, поэтому ограничение жёсткое).
- Каждый день должно работать не менее K станков (жёсткое ограничение).
- У каждого станка есть нежелательные дни работы, однако, если необходимо, этими пожеланиями можно пренебречь (мягкое ограничение).

1.2. Хромосома будет представлять из себя недельный план работы всех станков за каждую смену.

В нашей реализации хромосома будет представлена как трехмерный массив с размерностями N*D*S. Получаем, что алгоритм будет работать с хромосомой как с одномерным массивом, а результат мы получим в виде трёхмерного массива, что значительно упростит интерпретацию результата. Такое представление позволяет явно определить расписание для каждого станка, в том числе и расписание на конкретный день. Важно отметить, что элементы массива могут принимать только значения 0 и 1, следовательно, расписание станков представляется в битовом виде (0 – не работает, 1 - работает).

1.3. Для оценивания качества решения разработаем собственную метрику. Важно учесть, что выполнение жёстких ограничений более значимо, чем выполнение мягких ограничений.

Метрика будет представлена как счётчик всех отклонений от поставленных условий. Следовательно, чем ближе полученное значение метрики к нулю, тем лучше хромосома. Однако, если мы будем одинаково подсчитывать нарушения и мягких, и жёстких ограничений, невозможно будет соблюсти правило, что частое нарушение мягкого условия всегда лучше даже единичного нарушения жёсткого ограничения. Чтобы явно показать, когда были нарушены жёсткие ограничения, полученное число мягких условий будет делиться на N*D. Это гарантирует, что даже при полном несоблюдении мягких ограничений, значение метрики не превысит 1, если были соблюдены жёсткие ограничения. Реализация

разработанной метрики представлена в листинге 1.2.1. Вспомогательные функции можно видеть в листингах 1.2.2 и 1.2.3.

Листинг 1.2.1 – Функция приспособления

```
def fitness_function(individual):
      matrix = np.array(individual).reshape((N, D*S))
      hardLimit = 0
      softLimit = 0
      day_standbys = [0, 0, 0, 0, 0, 0, 0]

      # работа более трёх смен подряд
      for worker in matrix:
            hardLimit += overworking(worker)

      matrix = matrix.reshape((N, D, S))

      # минимальное число работающих станков
      for worker in matrix:
            for i in range(len(worker)):
                  day_standbys[i] += max(worker[i])
      for j in range(len(day_standbys)):
            if day_standbys[j] < K[j]:
                  hardLimit += 1

      # нежелательные дни работы
      for k in range(len(matrix)):
            softLimit += unwanted_days(matrix[k], k)

      return hardLimit + softLimit/(N*D)
```

Листинг 1.2.2 – Функция обнаружения переработок.

```
def overworking(worker):
      counter = 0
      limit = 0
      for el in worker:
```

```
5                if el == 1:
6                      counter += 1
7                elif el == 0 and counter > 3:
8                      limit += 1
9                      counter = 0
10               else:
11                     counter = 0
12         if counter > 3:
13               limit += 1
14         return limit
```

Листинг 1.2.3 – Функция обработки нежелательных дней.

```
1     def unwanted_days(worker, index):
2           limit = 0
3           for i in range(len(worker)):
4                if i+1 in UNWANTED_DAYS[index]:
5                      limit += max(worker[i])
6           return limit
```

2. Выполнение ГА.

2.1 Выберем операции для ГА.

Метод отбора: для данной задачи было принято решение использовать турнирный метод отбора. Этот подход попарно сравнивает решения и явно выбирает лучшее из двух рассматриваемых. Выбранный размер турнира: 3 - позволяет сохранить разнообразие решений и эффективно выбирает лучшее.

Метод скрещивания: было решено использовать метод двухточечного скрещивания, так как в нашей реализации хромосома представлена в виде набора 0 и 1, то есть имеет бинарную структуру. Данный метод позволяет значительно разнообразить потомство и хорошо сохраняет полезные комбинации.

Метод мутации: так как используемая нами хромосома представляет из себя набор бинарных значений, лучше всего подойдёт метод инвертирования

бита. При использовании этого метода каждый бит может быть изменён, что позволяет значительно разнообразить набор рассматриваемых комбинаций.

С помощью методов и структур библиотеки DEAP зададим необходимые операции для выполнения генетического алгоритма. Создание классов представлено в листинге 2.1.1. Регистрацию функций см. в листинге 2.1.2.

Листинг 2.1.1 – Создание классов.

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
```

В листинге 2.1.1 можно видеть создание двух классов: *FitnessMin* и *Individual*. Класс *FitnessMin* будет использоваться для оценки приспособленности особей. Класс *Individual* будет использоваться для представления особей, содержащих список генов и оценку приспособленности.

Листинг 2.1.2 – Регистрация функций.

```
toolbox = base.Toolbox()
toolbox.register("init_bool", random.randint, 0, 1)
toolbox.register("individualCreator", tools.initRepeat, creator.Individual, toolbox.init_bool, N*D*S)
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)

toolbox.register("evaluate", fitness_function)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.01)
```

2.2. Подберём параметры ГА.

Размер популяции и количество поколений: при подборе параметров запуска генетического алгоритма важно понимать, что для нашей задачи число комбинаций ограничено, и многие из них нарушают жёсткие ограничения, не говоря уже о мягких. Из этого следует, что не имеет смысла брать слишком большой начальный размер популяции (в результате тестов разных значений

было принято решение использовать значение в 50 особей). Также в результате ряда запусков было обнаружено, что для достижения значений метрики, близких к нулю, достаточно 200 поколений. Если уменьшить количество поколений, существует вероятность, что оптимальное решение не будет достигнуто. И наоборот, если число поколений увеличить, затраты на вычисление неоправданно увеличатся.

Вероятность скрещивания и мутации: в результате ряда запусков алгоритма с различными значениями вероятностей скрещивания и мутации было выяснено, что лучше всего алгоритм находил результат при вероятности скрещивания = 0.7 и вероятности мутации = 0.3.

В итоге имеем следующие параметры запуска:

Размер популяции: 50.

Количество поколений: 200.

Вероятность скрещивания: 0.7.

Вероятность мутации: 0.3.

2.3. Проведём оптимизацию. Для запуска алгоритма была написана функция *start()*, представленная в листинге 2.3.1.

Листинг 2.3.1 – Функция запуска ГА.

```
1     def start():
2           random.seed(64)
3           population = toolbox.populationCreator(n=POPULATION_SIZE)
4
5           stats = tools.Statistics(lambda ind: ind.fitness.values)
6           stats.register("min", np.min)
7           stats.register("avg", np.mean)
8
9           hof = tools.HallOfFame(HALL_OF_FAME_SIZE)
10
11          population, logbook = algorithms.eaSimple(population,
            toolbox, cxpb=P_CROSSOVER, mutpb=P_MUTATION,
            ngen=MAX_GENERATIONS, stats=stats, halloffame=hof,
            verbose=True)
```

```
12
13          return population, logbook, hof
```

В функции *start()* из листинга 2.3.1 сначала создаётся популяция размером *POPULATION_SIZE*. Далее осуществляется подготовка статистики, содержащей среднее и минимальное значения приспособленности популяции, и создание «зала славы», в котором будут храниться лучшие решения. После создания статистики запускается генетический алгоритм *eaSimple* с параметрами запуска, выбранными в п. 2.2.

Запустим функцию *start()* с заданными пользователем входными данными (см. листинг 2.3.2). Результат работы см. в листинге 2.3.3. Будут выведены только первые 5 и последние 5 поколений.

Листинг 2.3.2 – Входные данные.

```
1     POPULATION_SIZE = 50
2     P_CROSSOVER = 0.7
3     P_MUTATION = 0.3
4     MAX_GENERATIONS = 200
5     HALL_OF_FAME_SIZE = 10
6     N = 5
7     D = 7
8     S = 2
9     K = [4, 3, 4, 2, 5, 2, 3]
10    UNWANTED_DAYS = [[4, 7],[1, 3, 6],[2, 5],[3, 5],[2, 6]]
```

Листинг 2.3.3 – Результат работы алгоритма.

```
gen   nevals     min         avg
0     50         3.2         6.06171
1     40         2.22857     5.16914
2     41         2.22857     4.24971
3     42         1.17143     3.67771
4     39         2.17143     3.40571
…
196   42         0.0857143   0.148
197   42         0.0857143   0.165714
```

```
198   43          0.0857143  0.228
199   36          0.0857143  0.227429
200   43          0.0857143  0.246286
```

Из листинга 2.3.3 видно, что для этих входных данных не удалось полностью выполнить мягкое условие, однако, все жёсткие требования были соблюдены. Крайне маленькое значение характеристики min в 200 поколении говорит о том, что в подавляющем большинстве случаев мягкое ограничение соблюдается.

В результате работы алгоритма было получено лучшее решение, представленное в виде набора 0 и 1. В листинге 2.3.4 это решение преобразуется к легко читаемому табличному виду. Так как таблица получилась очень широкая, она будет представлена двумя рисунками – 2.3.1 и 2.3.2.

Листинг 2.3.4 – Функция преобразования особи к табличному виду.

```
1    def show_table(individual):
2          mytable = PrettyTable()
3          replacements = {1: 'Работает', 0: 'Не работает'}
4          schedule = [replacements.get(x, x) for x in individual]
5          schedule = np.array(schedule).reshape((N, D*S))
6          mytable.field_names = ["Номер станка", "Пн.Д", "Пн.В", "Вт.Д",
           "Вт.В", "Ср.Д", "Ср.В", "Чт.Д", "Чт.В", "Пт.Д", "Пт.В",
           "Сб.Д", "Сб.В", "Вс.Д", "Вс.В"]
7          for i in range(len(schedule)):
8                mytable.add_row(np.append([f'Станок {i+1}'],
                 schedule[i]))
9          print(mytable)
```

Номер станка	Пн.Д	Пн.В	Вт.Д	Вт.В	Ср.Д	Ср.В
Станок 1	Работает	Не работает	Работает	Не работает	Не работает	Работает
Станок 2	Не работает	Не работает	Работает	Работает	Не работает	Работает
Станок 3	Не работает	Работает	Не работает	Не работает	Работает	Не работает
Станок 4	Не работает	Работает	Работает	Работает	Не работает	Не работает
Станок 5	Не работает	Работает	Не работает	Не работает	Не работает	Работает

Рисунок 2.3.1 – Расписание работы станков в Пн-Ср.

Чт.Д	Чт.В	Пт.Д	Пт.В	Сб.Д	Сб.В	Вс.Д	Вс.В
Не работает	Не работает	Не работает	Работает	Не работает	Работает	Не работает	Не работает
Не работает	Работает	Работает	Не работает	Не работает	Не работает	Работает	Работает
Работает	Не работает	Не работает	Работает	Не работает	Работает	Работает	Работает
Не работает	Работает	Работает	Не работает	Работает	Не работает	Работает	Не работает
Не работает	Не работает	Не работает	Работает	Не работает	Не работает	Работает	Не работает

Рисунок 2.3.2 – Расписание работы станков в Чт-Вс.

На рисунках 2.3.1 и 2.3.2 можно видеть, что алгоритм успешно составляет расписание для всех станков.

2.4. Построим график зависимости минимальной и средней приспособленности в зависимости от поколения (см. листинг 2.4.1). Полученный график представлен на рисунке 2.4.1.

Листинг 2.4.1 – Рисование графика зависимости.

```
maxFitnessValues, meanFitnessValues = log.select("min", "avg")
sb.set_style("whitegrid")
plt.plot(maxFitnessValues, color='red')
plt.plot(meanFitnessValues, color='green')
plt.xlabel('Generation')
plt.ylabel('Min / Average Fitness')
plt.title('Min and Average Fitness over Generations')
plt.show()
```

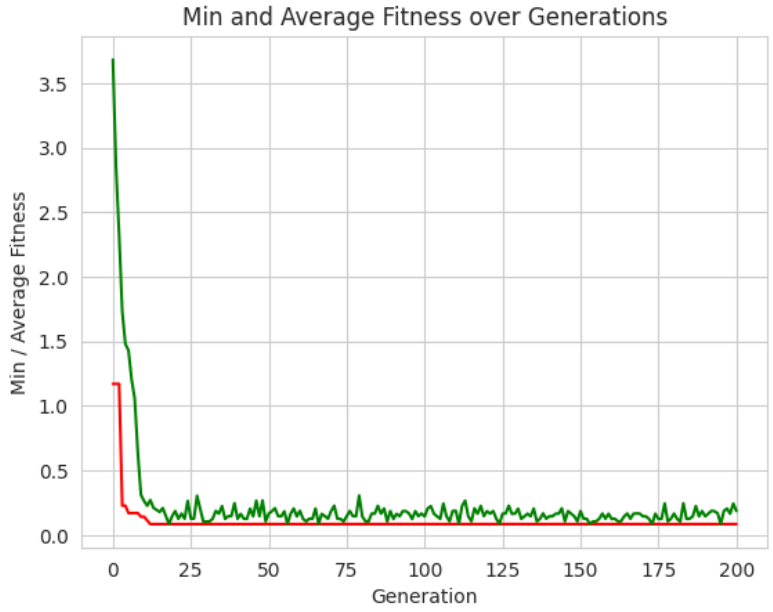

Рисунок 2.4.1 – График зависимости минимальной и средней приспособленности от поколения

На рисунке 2.4.1 видно, как минимальное значение приспособленности постепенно снижается с каждым поколением. Можно заметить, что минимальное значение приспособленности достигается уже к примерно 15 поколению, но при этом среднее значение продолжало колебаться на протяжении всей работы алгоритма, что моет быть связано с мутациями особей в каждом поколении.

Выводы.

В ходе выполнения лабораторной работы был изучен принцип работы генетического алгоритма, решена задача составления расписания с его использованием. В ходе решения задачи была разработана и реализована оценивающая метрика и изучены методы генетического алгоритма – отбор, скрещивание, мутация. Подобраны оптимальные для поставленной задачи параметры ГА.