Федеральное агентство связи

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

**Курсовая работа**

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент гр. БВТ2204

Альвицов Д.С.

Проверил:

ассистент кафедры Симонов С.Е.

Москва 2024

Содержание

[1 Цель работы 3](#__RefHeading___Toc936_2046167547)

[2 Выполнение работы 4](#__RefHeading___Toc938_2046167547)

[2.1 Импорты 4](#__RefHeading___Toc940_2046167547)

[2.2 Константы 4](#__RefHeading___Toc942_2046167547)

[2.4 Алгоритм Brute-Force 5](#__RefHeading___Toc944_2046167547)

[2.5 Генетический алгоритм 11](#__RefHeading___Toc946_2046167547)

[3 Вывод 19](#__RefHeading___Toc950_2046167547)

[4 Листинг кода 20](#__RefHeading___Toc952_2046167547)

[5. Список литературы 31](#__RefHeading___Toc1385_2046167547)

# 1 Цель работы

Разработка оптимального расписания движения автобусов с использованием генетического алгоритма и метода прямого перебора (в лоб) для минимизации количества водителей, максимизации прибыли и перевозки максимального количества людей

**Задача работы**

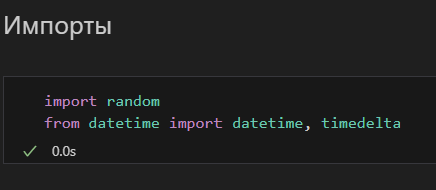
Максимизировать прибыль, учитывая зп водителей, поток людей, количество автобусов.

Условия:

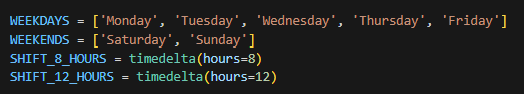
1. Разные водители могут работать в 2 варианта смен:
   1. работают по 8 часов в сутки, и у них выходной суббота и воскресенье, работают не ночью.
   2. работают по 12 часов в сутки, и 1 день работают, а 2 отдыхают, ночные смены подходят только для 12-часовых
2. каждый из них выбирает свои условия для перерыва на обед во время работы:
   1. когда автобус доезжает до конечной станции, то водитель берёт отдых в 1 час между 13 и 15 часами
   2. другой вид отдыха, когда водитель берёт перерывы по 10 или 15 минут каждые 2-4 часа, когда находится на конечной станции (само собой все водители одновременно не могут уйти на перерыв, т.к. автобусы всегда должны ходить по маршруту).
3. имеется пересменка 15 минут, когда одни водители сменяются другими. есть точка нулевого километра, где происходят отдых, пересменка и собственно это и есть конечная станция.
4. автобус должен ездить круглосуточно
5. на полный маршрут автобуса уходит например 1,5 часа, однако он не всегда выходит точное количество времени и в обычное время может уйти +/- 10 минут, и если задать час пик, то в это время добавляется ещё 5 минут.
6. те, кто работают по 8 часов должны выходить на работу в 6-10 часов утра
7. в конечном итоге нужно составить самое оптимальное расписание и сделать вывод во что-то типа таблички. В итоге Табличка кто когда выходит на работу кто когда обедает кто когда перерыв делает по дням

# 2 Выполнение работы

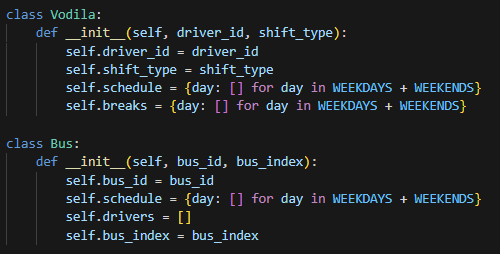
## 2.1 Импорты

Рисунок 1 — Импорты

## 2.2 Константы

Рисунок 2 — Константы

2.3 Объявление сущностей

Рисунок 3 — Объявление сущностей

Vodila – водитель

self.driver\_id – id водителя

self.shift\_type – вид смены (8 или 12 часов)

self.schedule – график работы

self.breaks – график перерывов

Bus – автобус

self.bus\_id – id автобуса

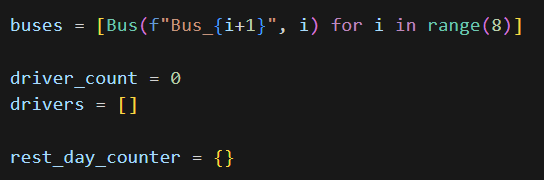
self.schedule – график эксплуатации

self.drivers – список водителей, которые работают именно на этом автобусе

self.bus\_index – индекс автобуса, нужен для корректной работы

## 2.4 Алгоритм Brute-Force

1) Локальные переменные

Рисунок 4 — Локальные переменные для Brute Force Алгоритма

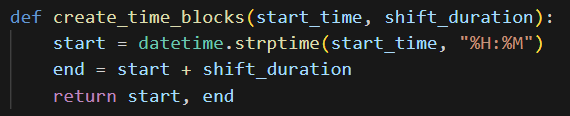
buses – список автобусов

driver\_count – количество занятых водителей

drivers – список с занятыми водителями

rest\_day\_counter – словарь с днями отдыха

2) Функция для создания рабочих смен

Рисунок 5 — Функция для создания рабочих смен для Brute Force Алгоритма

Аргументы функции :

start\_time – время начала рабочей смены

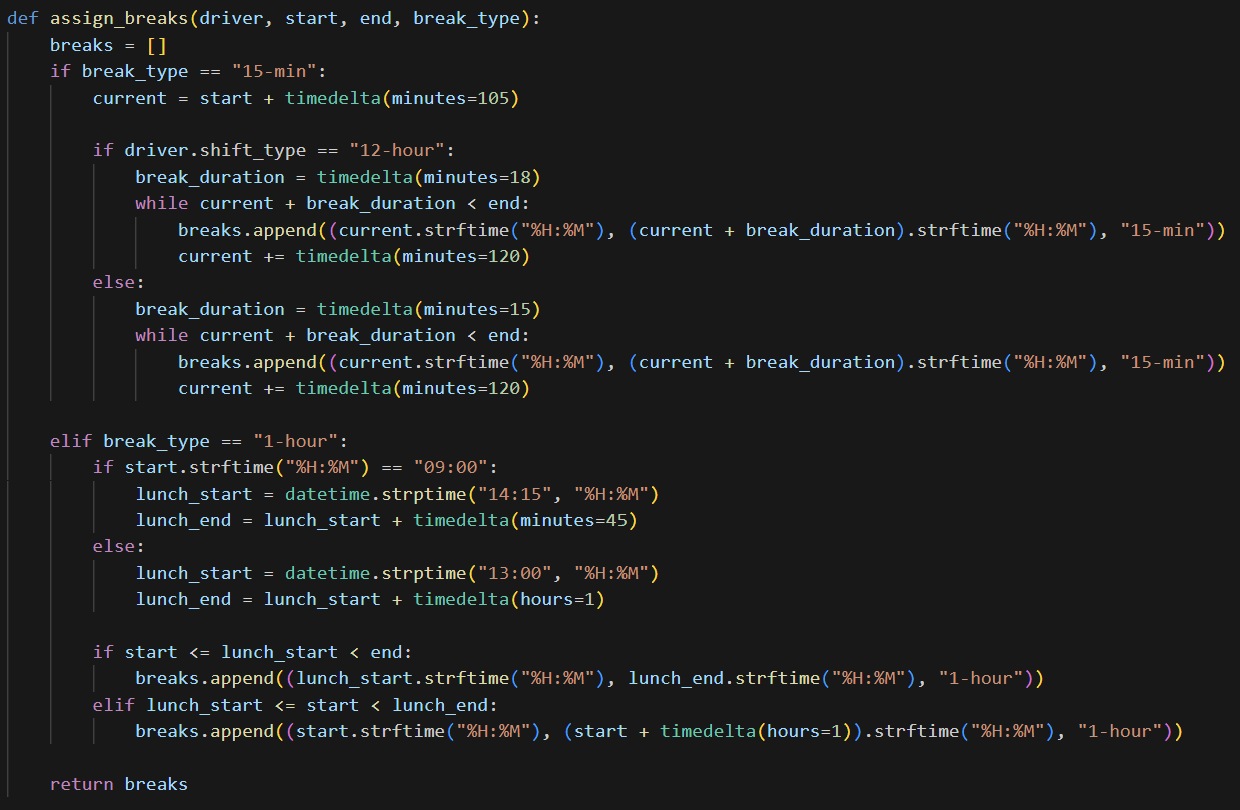
shift\_duration – длительность рабочей смены (8 или 12 часов)

Функция возвращает следующие значения:

start – начало рабочей смены

end – конец рабочей смены

3) Функция для заполнения перерывов

Рисунок 6 — Функция для заполнения перерывов для Brute Force Алгоритма

Аргументы функции:

driver – водитель, для которого осуществляется подбор перерывов

start – начало рабочей смены

end – конец рабочей смены

break\_type – вид перерыва (15 или 60 минут)

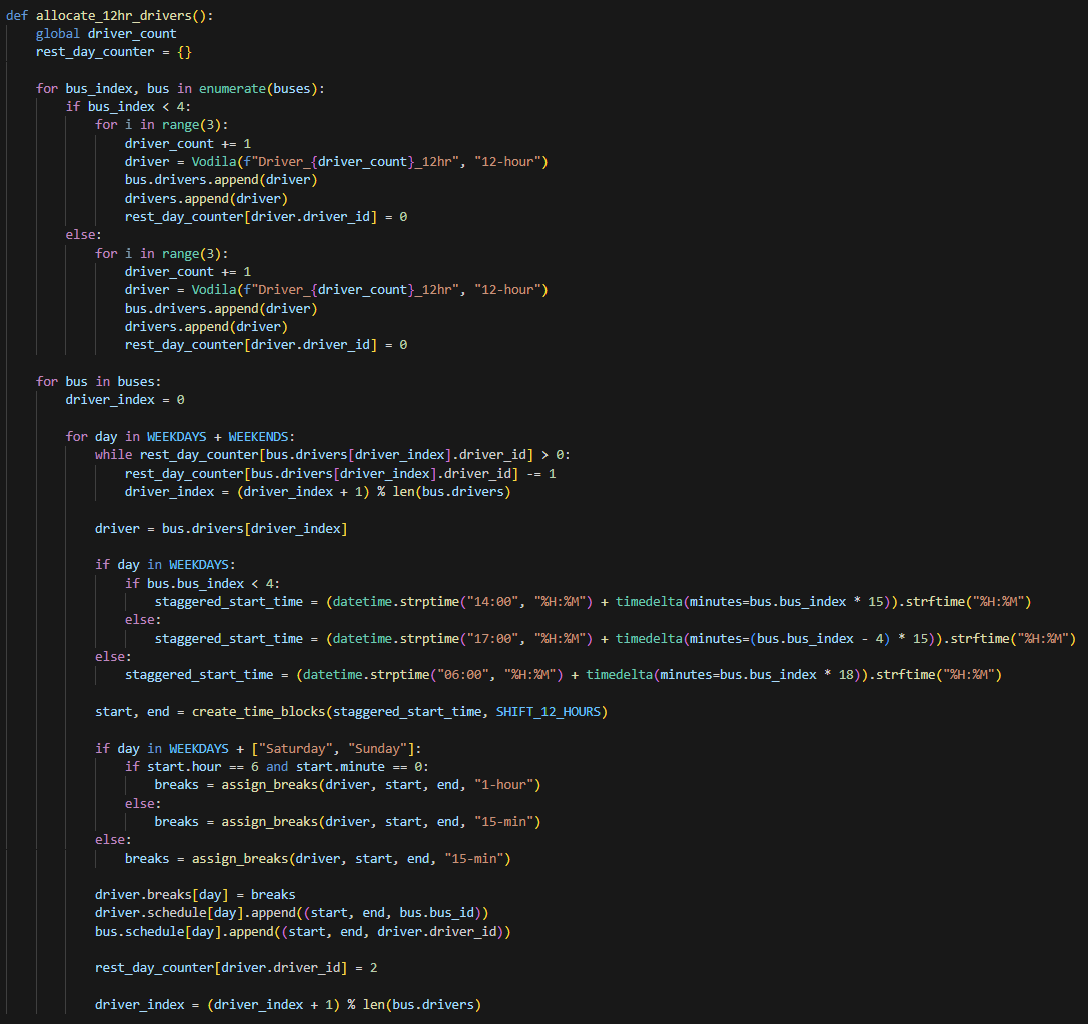
Функция возвращает следующие значения:

breaks – список с перерывами

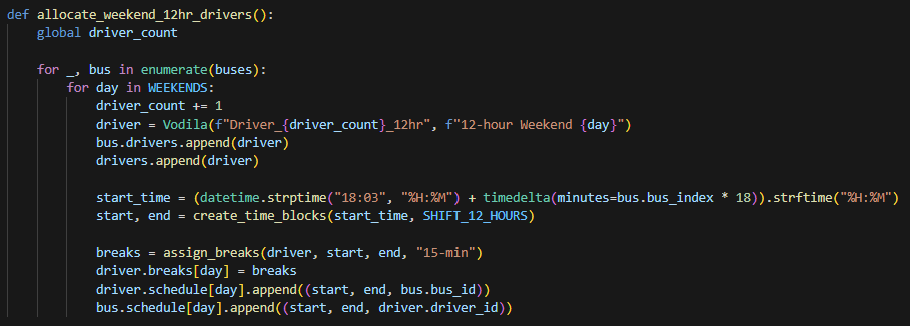
4) Функция для заполнения смен по 8 часов

Рисунок 7 — Функция для заполнения смен по 8 часов для Brute Force Алгоритма

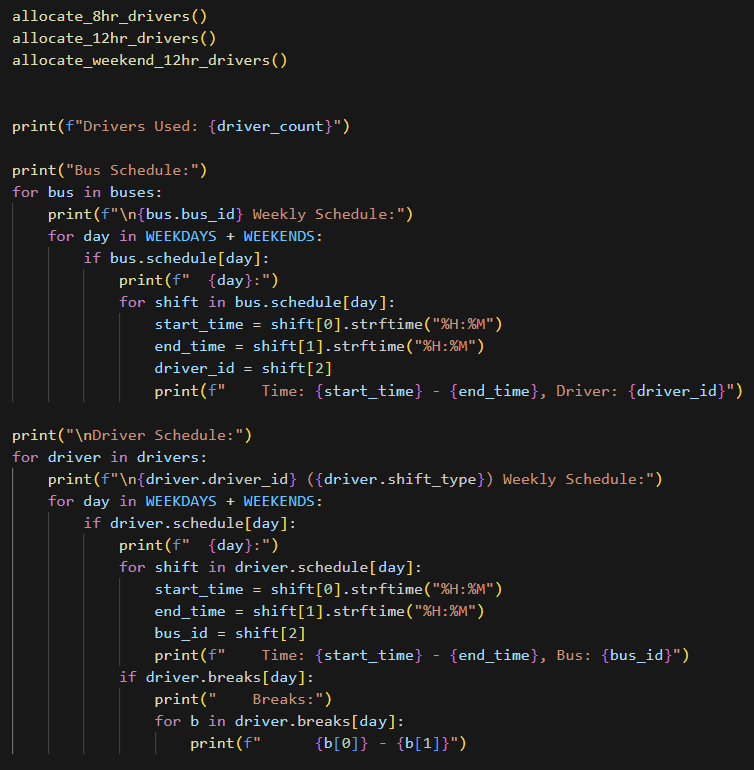
5) Функция для заполнения смен по 12 часов

Рисунок 8 - Функция для заполнения смен по 12 часов для Brute Force Алгоритма

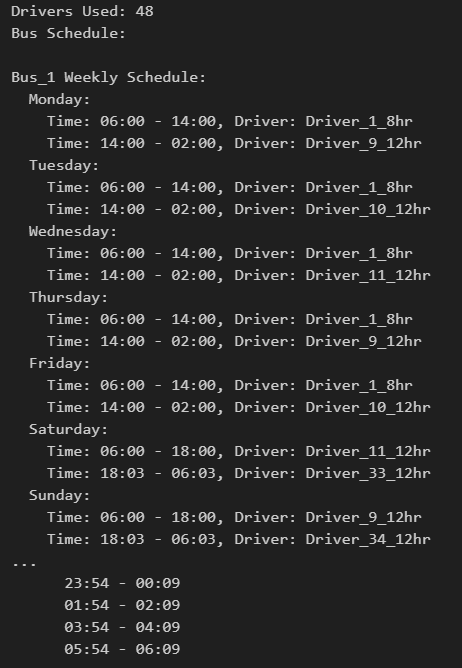
6) Функция для заполнения смен по 12 часов в выходные дни

Рисунок 9 - Функция для заполнения смен по 12 часов в выходные дни для Brute Force Алгоритма

7) Вызов функций и вывод результата

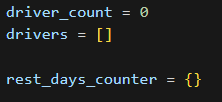
Рисунок 10 — Вызов функций и вывод результата для Brute-Force алгоритма

8) Результат работы Brute-Force алгоритма

Рисунок 11 — Результат работы Brute-Force алгоритма

## 2.5 Генетический алгоритм

1) Локальные переменные

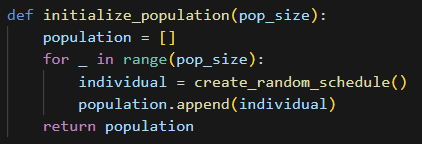
Рисунок 12 — Локальные переменные для генетического алгоритма

driver\_count – количество занятых водителей

drivers – список с занятыми водителями

rest\_day\_counter – словарь с днями отдыха

2) Функция для заполнения популяции

Рисунок 13 — Функция для заполнения популяции

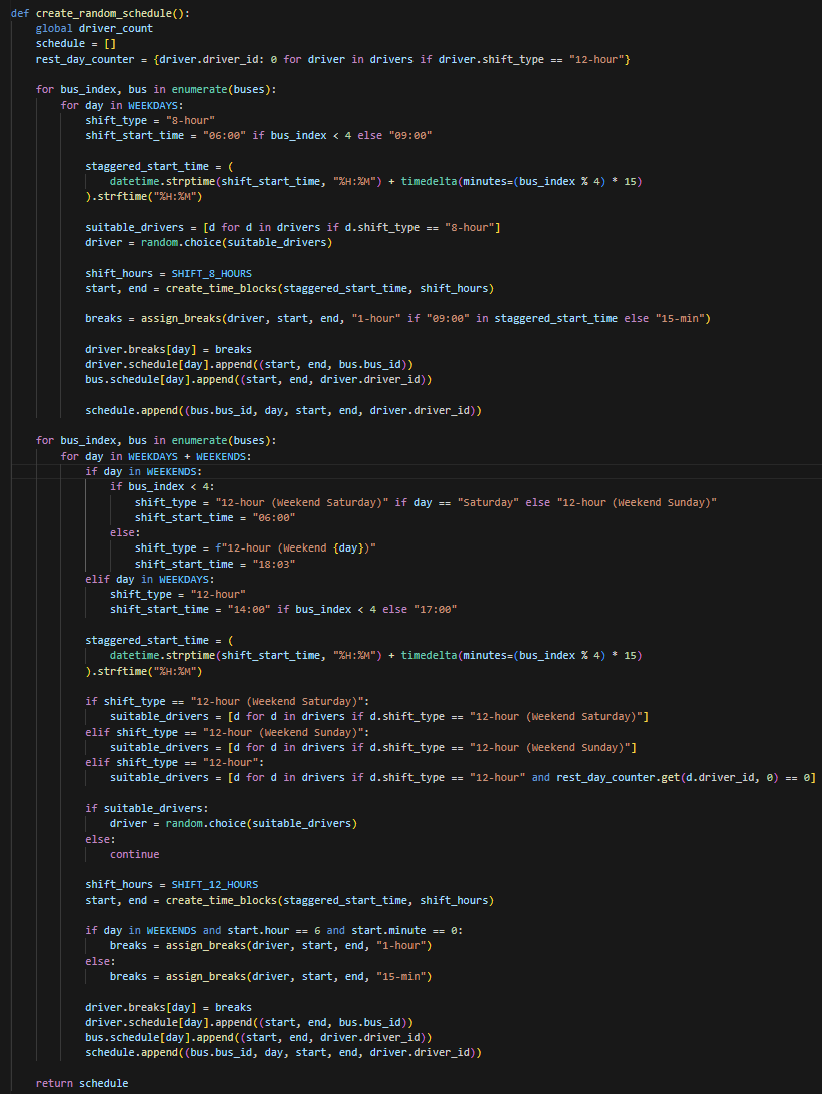
Аргументы функции:

pop\_size – количество индивидов в популяции

Функция возвращает следующие значения:

population – список с индивидами

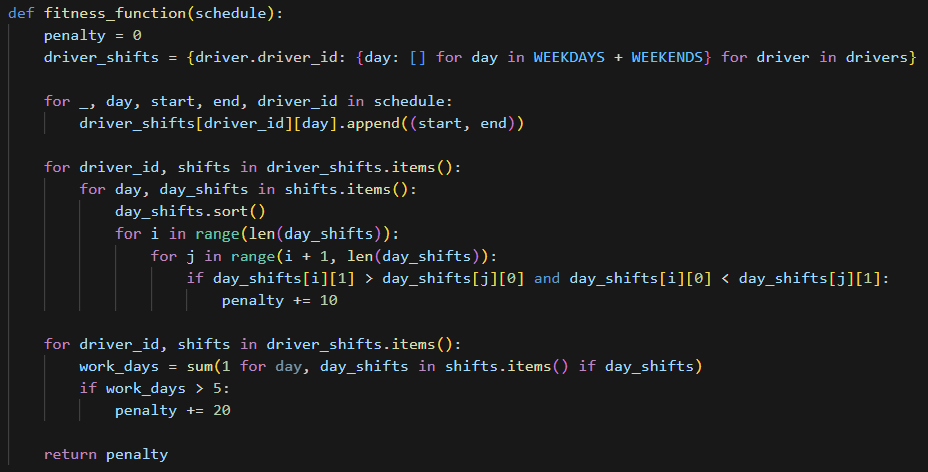
3) Функция для первичного заполнения расписания

Рисунок 14 — Функция для первичного заполнения расписания для генетического алгоритма

Функция возвращает следующие значения:

schedule – список с расписанием

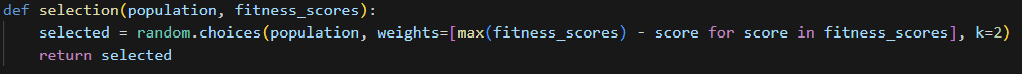
4) Функция для оценки качества алгоритма

Рисунок 15 — Функция для оценки качества алгоритма для генетического алгоритма

Функция возвращает следующие значения:

penalty — штраф (int), который является мерой того, насколько хорошо или плохо составлено расписание. Чем ниже значение штрафа, тем лучше расписание.

5) Функция для отбора особей

Рисунок 16 - Функция для отбора особей для генетического алгоритма

Функция выбирает менее приспособленных особей, чтобы разнообразить последующее поколение.

Аргументы функции:

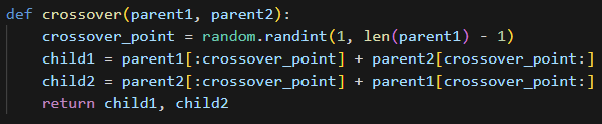
population – список индивидов

fitness\_scores – penalty

Функция возвращает следующие значения:

selected – список из двух выбранных особей

6) Функция для скрещивания генов

Рисунок 17 — Функция для скрещивания генов для генетического алгоритма

crossover\_point – случайная точка кроссовера.

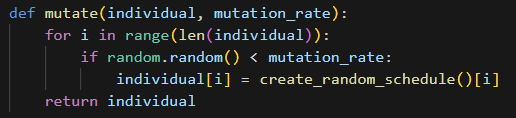
Аргументы функции:

parent1, parent2 – родители

Функция возвращает следующие значения:

child1, child2 – дети

7) Функция для мутации особи

Рисунок 18 — Функция для мутации особи для генетического алгоритма

Аргументы функции:

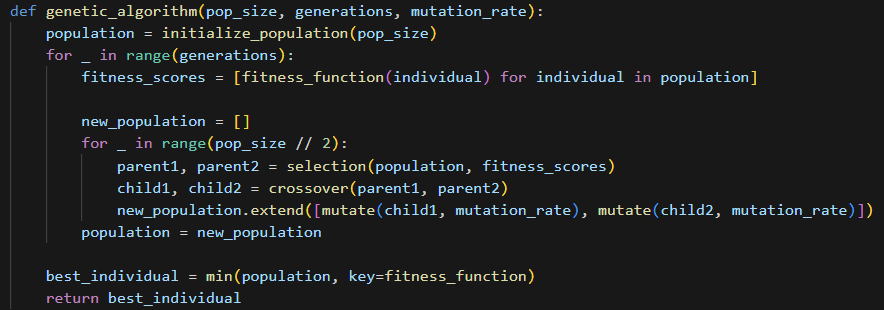
individual – особь, которая будет мутировать

mutation\_rate – вероятность мутации для каждого элемента в особи

Функция возвращает следующие значения:

individual – особь с мутациями

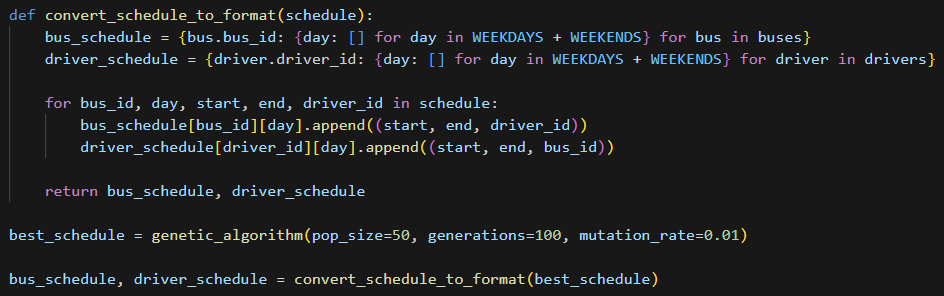
8) Функция реализации генетического алгоритма

Рисунок 19 — Функция реализации генетического алгоритма

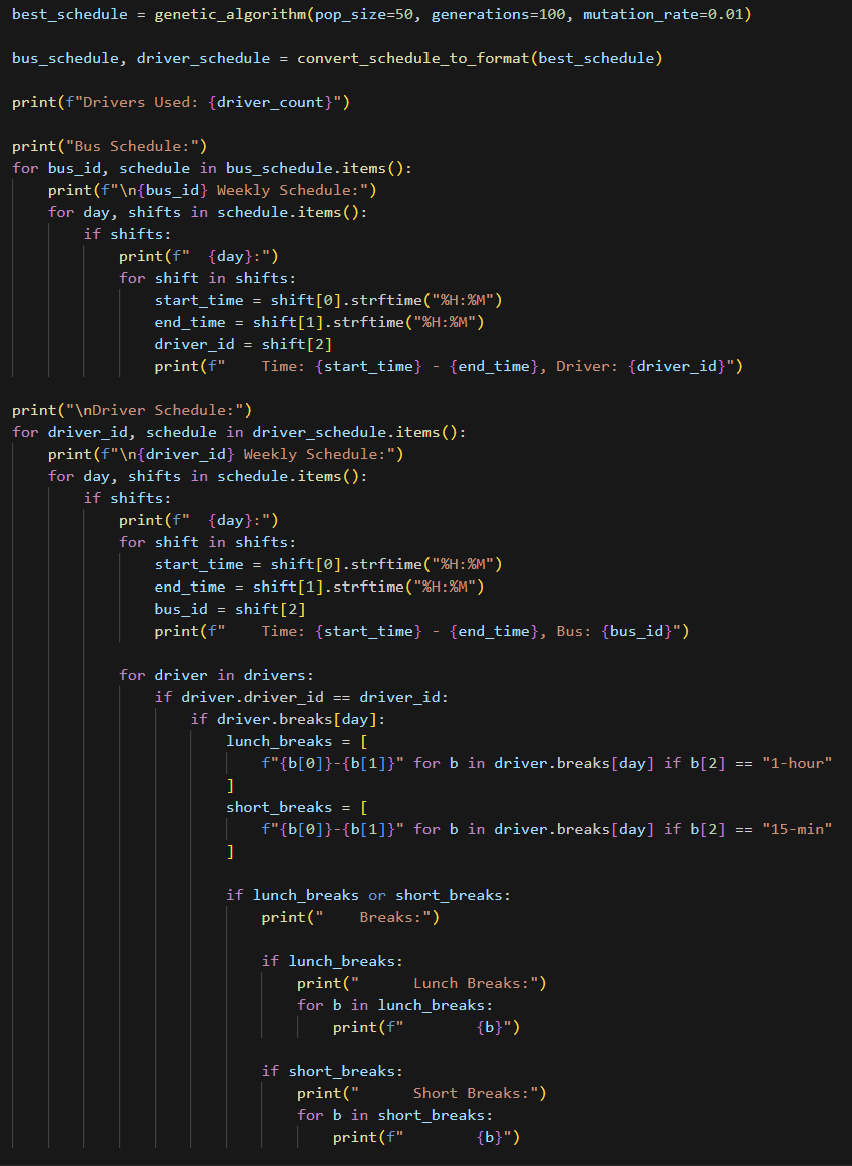
9) Функция для заполнения водителей

Рисунок 20 - Функция для заполнения водителей для генетического алгоритма

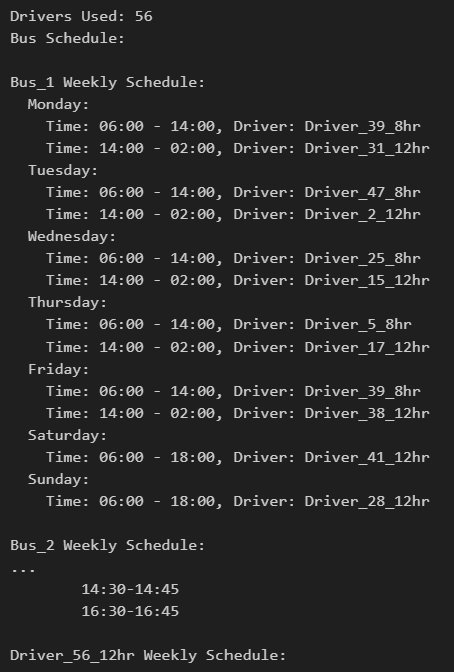
10) Функция для форматирования расписания для генетического алгоритма

Рисунок 21 - Функция для форматирования расписания для генетического алгоритма

11) Вызов функций и вывод результата

Рисунок 22 - Вызов функций и вывод результата для генетического алгоритма

12) Вывод

Рисунок 23 - Вывод

# 3 Вывод

В результате выполнения лабораторной работы были изучены методы регистрации двоичных сигналов. Уяснили методы стробирования и интегрирования.

# 4 Листинг кода

Курсовая\_СИАОД\_БВТ2204\_Альвицов\_Д.С.ipynb:

import random

from datetime import datetime, timedelta

WEEKDAYS = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

WEEKENDS = ['Saturday', 'Sunday']

SHIFT\_8\_HOURS = timedelta(hours=8)

SHIFT\_12\_HOURS = timedelta(hours=12)

class Vodila:

    def \_\_init\_\_(self, driver\_id, shift\_type):

        self.driver\_id = driver\_id

        self.shift\_type = shift\_type

        self.schedule = {day: [] for day in WEEKDAYS + WEEKENDS}

        self.breaks = {day: [] for day in WEEKDAYS + WEEKENDS}

class Bus:

    def \_\_init\_\_(self, bus\_id, bus\_index):

        self.bus\_id = bus\_id

        self.schedule = {day: [] for day in WEEKDAYS + WEEKENDS}

        self.drivers = []

        self.bus\_index = bus\_index

buses = [Bus(f"Bus\_{i+1}", i) for i in range(8)]

driver\_count = 0

drivers = []

rest\_day\_counter = {}

def create\_time\_blocks(start\_time, shift\_duration):

    start = datetime.strptime(start\_time, "%H:%M")

    end = start + shift\_duration

    return start, end

def assign\_breaks(driver, start, end, break\_type):

    breaks = []

    if break\_type == "15-min":

        current = start + timedelta(minutes=105)

        if driver.shift\_type == "12-hour":

            break\_duration = timedelta(minutes=18)

            while current + break\_duration < end:

                breaks.append((current.strftime("%H:%M"), (current + break\_duration).strftime("%H:%M"), "15-min"))

                current += timedelta(minutes=120)

        else:

            break\_duration = timedelta(minutes=15)

            while current + break\_duration < end:

                breaks.append((current.strftime("%H:%M"), (current + break\_duration).strftime("%H:%M"), "15-min"))

                current += timedelta(minutes=120)

    elif break\_type == "1-hour":

        if start.strftime("%H:%M") == "09:00":

            lunch\_start = datetime.strptime("14:15", "%H:%M")

            lunch\_end = lunch\_start + timedelta(minutes=45)

        else:

            lunch\_start = datetime.strptime("13:00", "%H:%M")

            lunch\_end = lunch\_start + timedelta(hours=1)

        if start <= lunch\_start < end:

            breaks.append((lunch\_start.strftime("%H:%M"), lunch\_end.strftime("%H:%M"), "1-hour"))

        elif lunch\_start <= start < lunch\_end:

            breaks.append((start.strftime("%H:%M"), (start + timedelta(hours=1)).strftime("%H:%M"), "1-hour"))

    return breaks

def allocate\_8hr\_drivers():

    global driver\_count

    for bus\_index, bus in enumerate(buses):

        if bus\_index < 4:

            shift\_start\_time = "06:00"

        else:

            shift\_start\_time = "09:00"

        driver\_count += 1

        driver = Vodila(f"Driver\_{driver\_count}\_8hr", "8-hour")

        drivers.append(driver)

        for day in WEEKDAYS:

            staggered\_start\_time = (

                datetime.strptime(shift\_start\_time, "%H:%M") + timedelta(minutes=(bus\_index % 4) \* 15)

            ).strftime("%H:%M")

            start, end = create\_time\_blocks(staggered\_start\_time, SHIFT\_8\_HOURS)

            if "09:00" in staggered\_start\_time:

                breaks = assign\_breaks(driver, start, end, "1-hour")

            else:

                breaks = assign\_breaks(driver, start, end, "15-min")

            driver.breaks[day] = breaks

            driver.schedule[day].append((start, end, bus.bus\_id))

            bus.schedule[day].append((start, end, driver.driver\_id))

def allocate\_12hr\_drivers():

    global driver\_count

    rest\_day\_counter = {}

    for bus\_index, bus in enumerate(buses):

        if bus\_index < 4:

            for i in range(3):

                driver\_count += 1

                driver = Vodila(f"Driver\_{driver\_count}\_12hr", "12-hour")

                bus.drivers.append(driver)

                drivers.append(driver)

                rest\_day\_counter[driver.driver\_id] = 0

        else:

            for i in range(3):

                driver\_count += 1

                driver = Vodila(f"Driver\_{driver\_count}\_12hr", "12-hour")

                bus.drivers.append(driver)

                drivers.append(driver)

                rest\_day\_counter[driver.driver\_id] = 0

    for bus in buses:

        driver\_index = 0

        for day in WEEKDAYS + WEEKENDS:

            while rest\_day\_counter[bus.drivers[driver\_index].driver\_id] > 0:

                rest\_day\_counter[bus.drivers[driver\_index].driver\_id] -= 1

                driver\_index = (driver\_index + 1) % len(bus.drivers)

            driver = bus.drivers[driver\_index]

            if day in WEEKDAYS:

                if bus.bus\_index < 4:

                    staggered\_start\_time = (datetime.strptime("14:00", "%H:%M") + timedelta(minutes=bus.bus\_index \* 15)).strftime("%H:%M")

                else:

                    staggered\_start\_time = (datetime.strptime("17:00", "%H:%M") + timedelta(minutes=(bus.bus\_index - 4) \* 15)).strftime("%H:%M")

            else:

                staggered\_start\_time = (datetime.strptime("06:00", "%H:%M") + timedelta(minutes=bus.bus\_index \* 18)).strftime("%H:%M")

            start, end = create\_time\_blocks(staggered\_start\_time, SHIFT\_12\_HOURS)

            if day in WEEKDAYS + ["Saturday", "Sunday"]:

                if start.hour == 6 and start.minute == 0:

                    breaks = assign\_breaks(driver, start, end, "1-hour")

                else:

                    breaks = assign\_breaks(driver, start, end, "15-min")

            else:

                breaks = assign\_breaks(driver, start, end, "15-min")

            driver.breaks[day] = breaks

            driver.schedule[day].append((start, end, bus.bus\_id))

            bus.schedule[day].append((start, end, driver.driver\_id))

            rest\_day\_counter[driver.driver\_id] = 2

            driver\_index = (driver\_index + 1) % len(bus.drivers)

def allocate\_weekend\_12hr\_drivers():

    global driver\_count

    for \_, bus in enumerate(buses):

        for day in WEEKENDS:

            driver\_count += 1

            driver = Vodila(f"Driver\_{driver\_count}\_12hr", f"12-hour Weekend {day}")

            bus.drivers.append(driver)

            drivers.append(driver)

            start\_time = (datetime.strptime("18:03", "%H:%M") + timedelta(minutes=bus.bus\_index \* 18)).strftime("%H:%M")

            start, end = create\_time\_blocks(start\_time, SHIFT\_12\_HOURS)

            breaks = assign\_breaks(driver, start, end, "15-min")

            driver.breaks[day] = breaks

            driver.schedule[day].append((start, end, bus.bus\_id))

            bus.schedule[day].append((start, end, driver.driver\_id))

allocate\_8hr\_drivers()

allocate\_12hr\_drivers()

allocate\_weekend\_12hr\_drivers()

print(f"Drivers Used: {driver\_count}")

print("Bus Schedule:")

for bus in buses:

    print(f"\n{bus.bus\_id} Weekly Schedule:")

    for day in WEEKDAYS + WEEKENDS:

        if bus.schedule[day]:

            print(f"  {day}:")

            for shift in bus.schedule[day]:

                start\_time = shift[0].strftime("%H:%M")

                end\_time = shift[1].strftime("%H:%M")

                driver\_id = shift[2]

                print(f"    Time: {start\_time} - {end\_time}, Driver: {driver\_id}")

print("\nDriver Schedule:")

for driver in drivers:

    print(f"\n{driver.driver\_id} ({driver.shift\_type}) Weekly Schedule:")

    for day in WEEKDAYS + WEEKENDS:

        if driver.schedule[day]:

            print(f"  {day}:")

            for shift in driver.schedule[day]:

                start\_time = shift[0].strftime("%H:%M")

                end\_time = shift[1].strftime("%H:%M")

                bus\_id = shift[2]

                print(f"    Time: {start\_time} - {end\_time}, Bus: {bus\_id}")

            if driver.breaks[day]:

                print("    Breaks:")

                for b in driver.breaks[day]:

                    print(f"      {b[0]} - {b[1]}")

driver\_count = 0

drivers = []

rest\_days\_counter = {}

def initialize\_population(pop\_size):

    population = []

    for \_ in range(pop\_size):

        individual = create\_random\_schedule()

        population.append(individual)

    return population

def create\_random\_schedule():

    global driver\_count

    schedule = []

    rest\_day\_counter = {driver.driver\_id: 0 for driver in drivers if driver.shift\_type == "12-hour"}

    for bus\_index, bus in enumerate(buses):

        for day in WEEKDAYS:

            shift\_type = "8-hour"

            shift\_start\_time = "06:00" if bus\_index < 4 else "09:00"

            staggered\_start\_time = (

                datetime.strptime(shift\_start\_time, "%H:%M") + timedelta(minutes=(bus\_index % 4) \* 15)

            ).strftime("%H:%M")

            suitable\_drivers = [d for d in drivers if d.shift\_type == "8-hour"]

            driver = random.choice(suitable\_drivers)

            shift\_hours = SHIFT\_8\_HOURS

            start, end = create\_time\_blocks(staggered\_start\_time, shift\_hours)

            breaks = assign\_breaks(driver, start, end, "1-hour" if "09:00" in staggered\_start\_time else "15-min")

            driver.breaks[day] = breaks

            driver.schedule[day].append((start, end, bus.bus\_id))

            bus.schedule[day].append((start, end, driver.driver\_id))

            schedule.append((bus.bus\_id, day, start, end, driver.driver\_id))

    for bus\_index, bus in enumerate(buses):

        for day in WEEKDAYS + WEEKENDS:

            if day in WEEKENDS:

                if bus\_index < 4:

                    shift\_type = "12-hour (Weekend Saturday)" if day == "Saturday" else "12-hour (Weekend Sunday)"

                    shift\_start\_time = "06:00"

                else:

                    shift\_type = f"12-hour (Weekend {day})"

                    shift\_start\_time = "18:03"

            elif day in WEEKDAYS:

                shift\_type = "12-hour"

                shift\_start\_time = "14:00" if bus\_index < 4 else "17:00"

            staggered\_start\_time = (

                datetime.strptime(shift\_start\_time, "%H:%M") + timedelta(minutes=(bus\_index % 4) \* 15)

            ).strftime("%H:%M")

            if shift\_type == "12-hour (Weekend Saturday)":

                suitable\_drivers = [d for d in drivers if d.shift\_type == "12-hour (Weekend Saturday)"]

            elif shift\_type == "12-hour (Weekend Sunday)":

                suitable\_drivers = [d for d in drivers if d.shift\_type == "12-hour (Weekend Sunday)"]

            elif shift\_type == "12-hour":

                suitable\_drivers = [d for d in drivers if d.shift\_type == "12-hour" and rest\_day\_counter.get(d.driver\_id, 0) == 0]

            if suitable\_drivers:

                driver = random.choice(suitable\_drivers)

            else:

                continue

            shift\_hours = SHIFT\_12\_HOURS

            start, end = create\_time\_blocks(staggered\_start\_time, shift\_hours)

            if day in WEEKENDS and start.hour == 6 and start.minute == 0:

                breaks = assign\_breaks(driver, start, end, "1-hour")

            else:

                breaks = assign\_breaks(driver, start, end, "15-min")

            driver.breaks[day] = breaks

            driver.schedule[day].append((start, end, bus.bus\_id))

            bus.schedule[day].append((start, end, driver.driver\_id))

            schedule.append((bus.bus\_id, day, start, end, driver.driver\_id))

    return schedule

def fitness\_function(schedule):

    penalty = 0

    driver\_shifts = {driver.driver\_id: {day: [] for day in WEEKDAYS + WEEKENDS} for driver in drivers}

    for \_, day, start, end, driver\_id in schedule:

        driver\_shifts[driver\_id][day].append((start, end))

    for driver\_id, shifts in driver\_shifts.items():

        for day, day\_shifts in shifts.items():

            day\_shifts.sort()

            for i in range(len(day\_shifts)):

                for j in range(i + 1, len(day\_shifts)):

                    if day\_shifts[i][1] > day\_shifts[j][0] and day\_shifts[i][0] < day\_shifts[j][1]:

                        penalty += 10

    for driver\_id, shifts in driver\_shifts.items():

        work\_days = sum(1 for day, day\_shifts in shifts.items() if day\_shifts)

        if work\_days > 5:

            penalty += 20

    return penalty

def selection(population, fitness\_scores):

    selected = random.choices(population, weights=[max(fitness\_scores) - score for score in fitness\_scores], k=2)

    return selected

def crossover(parent1, parent2):

    crossover\_point = random.randint(1, len(parent1) - 1)

    child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

    child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

    return child1, child2

def mutate(individual, mutation\_rate):

    for i in range(len(individual)):

        if random.random() < mutation\_rate:

            individual[i] = create\_random\_schedule()[i]

    return individual

def genetic\_algorithm(pop\_size, generations, mutation\_rate):

    population = initialize\_population(pop\_size)

    for \_ in range(generations):

        fitness\_scores = [fitness\_function(individual) for individual in population]

        new\_population = []

        for \_ in range(pop\_size // 2):

            parent1, parent2 = selection(population, fitness\_scores)

            child1, child2 = crossover(parent1, parent2)

            new\_population.extend([mutate(child1, mutation\_rate), mutate(child2, mutation\_rate)])

        population = new\_population

    best\_individual = min(population, key=fitness\_function)

    return best\_individual

def allocate\_drivers():

    global driver\_count

    for \_, bus in enumerate(buses):

        for \_ in range(3):

            driver\_count += 1

            driver = Vodila(f"Driver\_{driver\_count}\_12hr", "12-hour")

            bus.drivers.append(driver)

            drivers.append(driver)

        for \_ in range(2):

            driver\_count += 1

            driver = Vodila(f"Driver\_{driver\_count}\_8hr", "8-hour")

            bus.drivers.append(driver)

            drivers.append(driver)

        driver\_count += 1

        driver = Vodila(f"Driver\_{driver\_count}\_12hr", "12-hour (Weekend Saturday)")

        bus.drivers.append(driver)

        drivers.append(driver)

        driver\_count += 1

        driver = Vodila(f"Driver\_{driver\_count}\_12hr", "12-hour (Weekend Sunday)")

        bus.drivers.append(driver)

        drivers.append(driver)

allocate\_drivers()

def convert\_schedule\_to\_format(schedule):

    bus\_schedule = {bus.bus\_id: {day: [] for day in WEEKDAYS + WEEKENDS} for bus in buses}

    driver\_schedule = {driver.driver\_id: {day: [] for day in WEEKDAYS + WEEKENDS} for driver in drivers}

    for bus\_id, day, start, end, driver\_id in schedule:

        bus\_schedule[bus\_id][day].append((start, end, driver\_id))

        driver\_schedule[driver\_id][day].append((start, end, bus\_id))

    return bus\_schedule, driver\_schedule

best\_schedule = genetic\_algorithm(pop\_size=50, generations=100, mutation\_rate=0.01)

bus\_schedule, driver\_schedule = convert\_schedule\_to\_format(best\_schedule)

print(f"Drivers Used: {driver\_count}")

print("Bus Schedule:")

for bus\_id, schedule in bus\_schedule.items():

    print(f"\n{bus\_id} Weekly Schedule:")

    for day, shifts in schedule.items():

        if shifts:

            print(f"  {day}:")

            for shift in shifts:

                start\_time = shift[0].strftime("%H:%M")

                end\_time = shift[1].strftime("%H:%M")

                driver\_id = shift[2]

                print(f"    Time: {start\_time} - {end\_time}, Driver: {driver\_id}")

print("\nDriver Schedule:")

for driver\_id, schedule in driver\_schedule.items():

    print(f"\n{driver\_id} Weekly Schedule:")

    for day, shifts in schedule.items():

        if shifts:

            print(f"  {day}:")

            for shift in shifts:

                start\_time = shift[0].strftime("%H:%M")

                end\_time = shift[1].strftime("%H:%M")

                bus\_id = shift[2]

                print(f"    Time: {start\_time} - {end\_time}, Bus: {bus\_id}")

            for driver in drivers:

                if driver.driver\_id == driver\_id:

                    if driver.breaks[day]:

                        lunch\_breaks = [

                            f"{b[0]}-{b[1]}" for b in driver.breaks[day] if b[2] == "1-hour"

                        ]

                        short\_breaks = [

                            f"{b[0]}-{b[1]}" for b in driver.breaks[day] if b[2] == "15-min"

                        ]

                        if lunch\_breaks or short\_breaks:

                            print("    Breaks:")

                            if lunch\_breaks:

                                print("      Lunch Breaks:")

                                for b in lunch\_breaks:

                                    print(f"        {b}")

                            if short\_breaks:

                                print("      Short Breaks:")

                                for b in short\_breaks:

                                    print(f"        {b}")

# **5. Список литературы**

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - Питер: 2022. - 288 с.