МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский Технический Университет Связи И Информатики (MTUCI)»**

Кафедра «Математическая кибернетика и информационные технологии»

**Курсовая Работа**

по дисциплине «**Структуры и алгоритмы обработки данных**»

Выполнил:

студент 3 курса гр. БВТ2204

Ибрагимов Т.Э.

Москва 2024 г.

**Содержание**

[Цель 3](#_Toc184743264)

[Задача работы 3](#_Toc184743265)

[Реализация 4](#_Toc184743266)

[В Лоб 4](#_Toc184743266)

[Генетический алгоритм 8](#_Toc184743273)

[Сравнение алгоритмов 15](#_Toc184743274)

[Итог 15](#_Toc184743274)

[Список источников 16](#_Toc184743274)

[Листинг кода 17](#_Toc184743275)

**Цель**

Разработка расписания движения автобусов с использованием метода прямого перебора, то есть в лоб для оптимального расписания работы водителей с учетом всех ограничений, а также с помощью генетического алгоритма и обеспечить равномерное распределение смен и соблюдений правил по отдыху.

**Задача работы**

1. Разработка оптимального расписания работы водителей автобусов

а) Создать расписание для 8 водителей, учитывая специфику их смен

б) Учитывать временные ограничения, такие как продолжительность смен и допустимые часы работы.

1. Учёт ограничений и распределение ресурсов

а) Убедиться, что каждый автобус обслуживается не более чем одним водителем в каждый час.

б) Гарантировать, что водители работают строго свои смены (8 или 12 часов подряд) без перекрытия.

1. Распределение выходных дней и смен

а) Организовать выходные для водителей с короткими сменами в субботу и воскресенье.

б) Разработать график для водителей с длинными сменами, чтобы чередовать их работу через каждые три дня.

**Реализация**

Brute Force

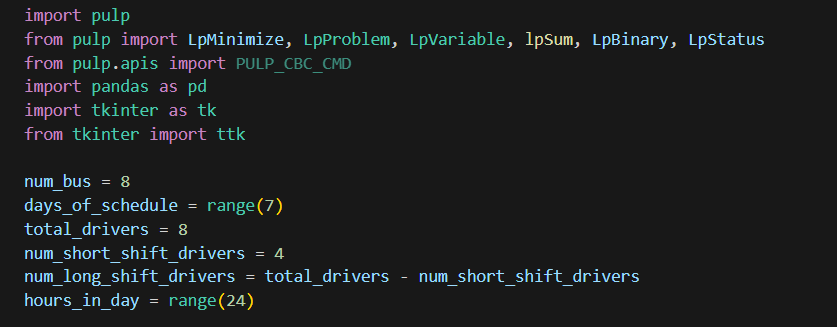
****

Рис 1 – Импорт библиотек определение требования для формирования расписания

Импортируются необходимые библиотеки:

Pulp **-** используется для создания и решения задачи оптимизации

Pandas - для формирования таблицы с расписанием

Tkinter - для создания графического интерфейса (визуализация расписания)

Создание задачи оптимизации:

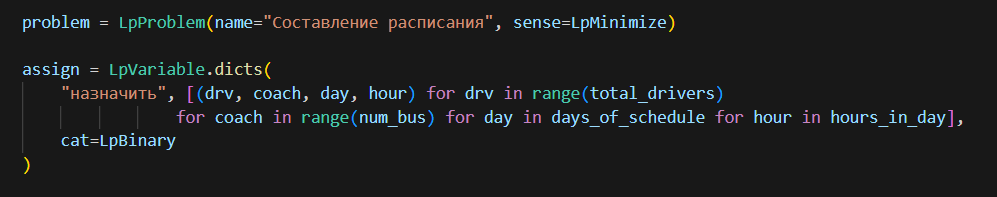


Рис 2 – Задача оптимизации

**assign** — словарь переменных. Каждая переменная имеет значение 0 или 1 (тип **LpBinary**)

Создаётся задача оптимизации, то есть problem , что определяет минимизацию времени, потраченного на решение задачи

Массив переменных assign определяет, будет ли водитель работать на автобусе в конкретный день и час

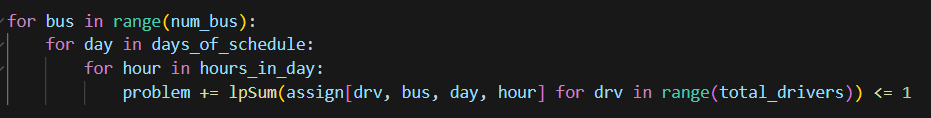


Рис 3 – Ограничение один водитель значит один автобус

Тут оно гарантирует, что на одном автобусе в один час может работать только один водитель

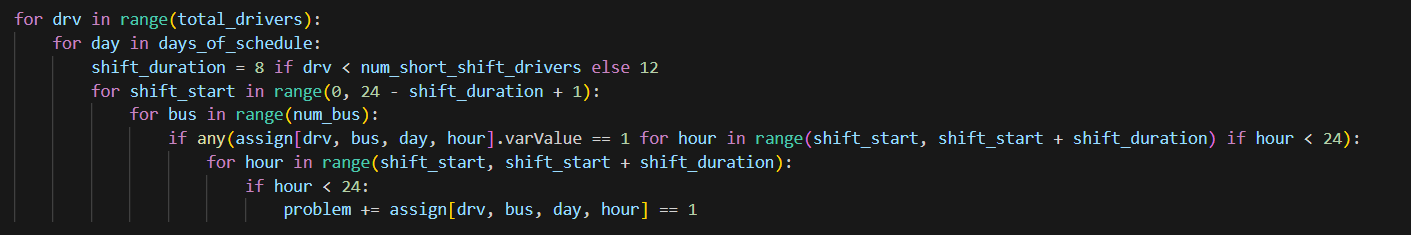


Рис 4 – Соблюдение длительности смен

У нас короткая смена – 8 часов

А для длинной – 12 часов

Здесь оно формирует ограничение на продолжительность смены и её последовательность

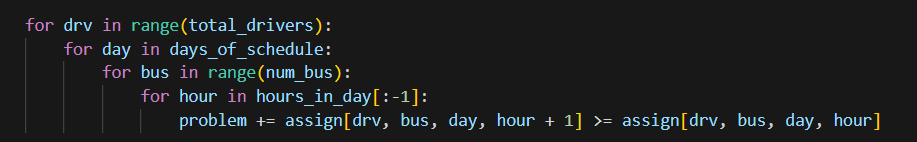


Рис 5 – Последовательность работы

Устанавливается непрерывность смены: если водитель работает в один час, то он также должен работать в следующий час, пока смена не завершится

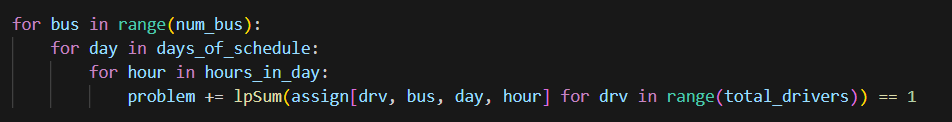
****

Рис 6 - Все автобусы заняты

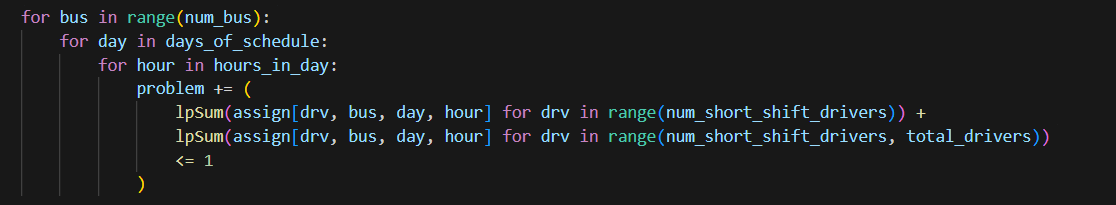
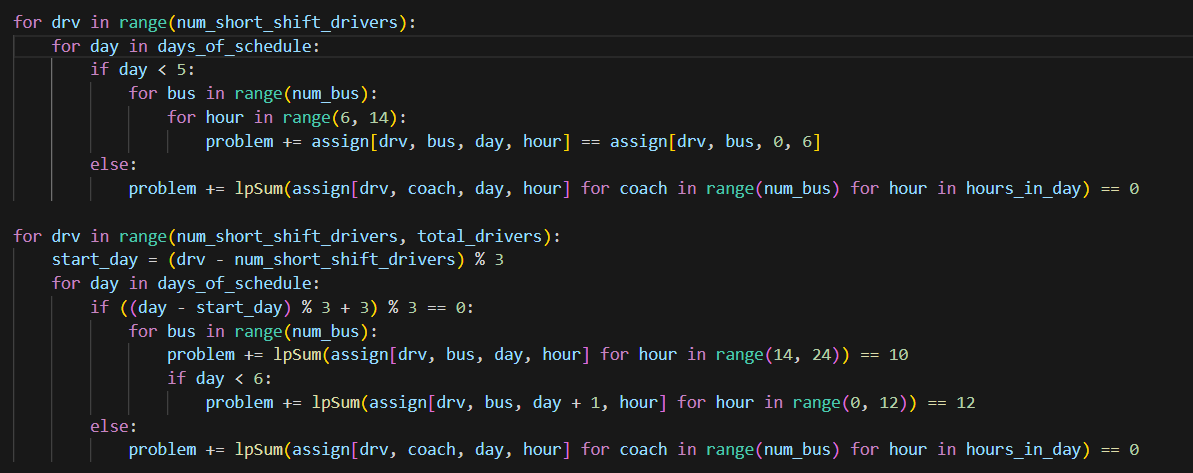


Рис 7 – Еще одно ограничение

Отвечает за **ограничение** на количество водителей, которые одновременно работают на одном автобусе в определённый момент времени



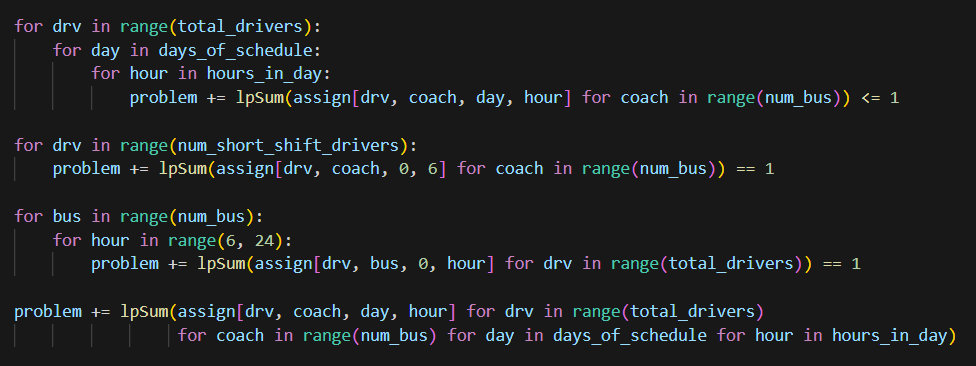


Рис 8,9 – Выходные

Здесь создаются ограничения для каждого автобуса в каждый час каждого дня.

Например, для автобуса 1 в понедельник в 10:00 код гарантирует, что за рулем не будет более одного водителя.

Затем суммируются **все короткосменные водители**, работающие в этот час на этом автобусе

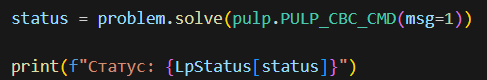


Рис. 10 – Решение задачи оптимизации

**Функция solve** запускает процесс решения задачи линейного программирования. Используется CBC - это встроенный решатель линейных программ, поддерживаемый PuLP.

msg = 1 означает вывод в консоль отладочного сообщения о процессе решения задачи. После этого возвращается статус, который показывает как завершилось решение задачи

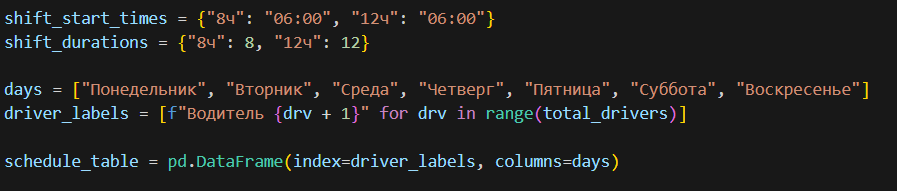
Может быть три случая:

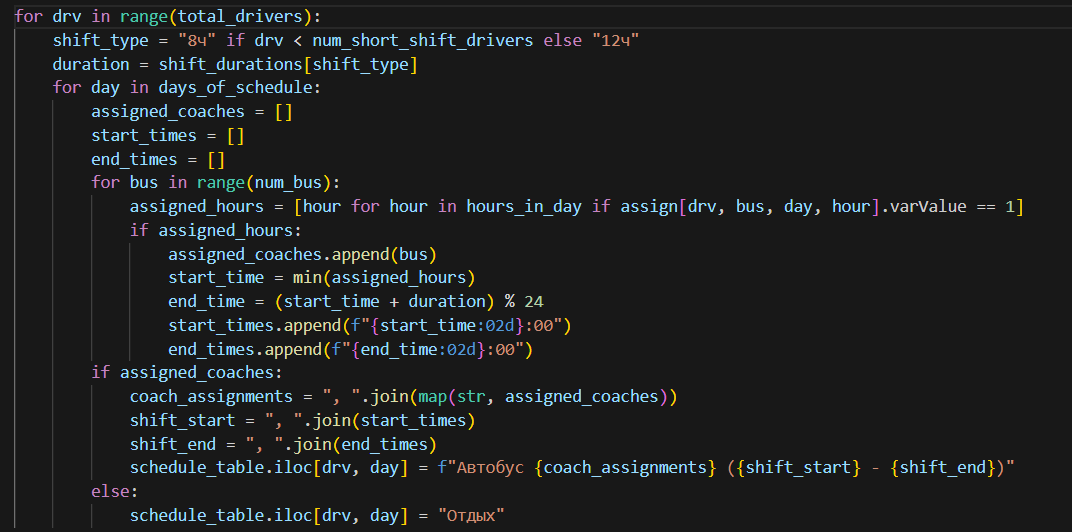
Optimal - Решение успешно найдено, и оно оптимально в рамках поставленной задачи

Infeasible - Решение невозможно из-за слишком жестких ограничений

Underfined - Решение не было найдено

Визуальная часть





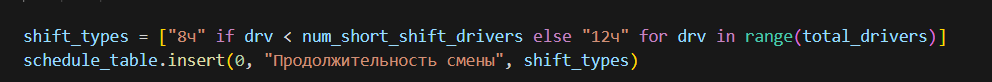


Рис 11,12,13 – Формирование таблицы расписания

То есть наши результаты конвертируются в удобный формат для визуализации



Рис 14 – Создание таблички

Создаем визуальную таблицу с нашими данные

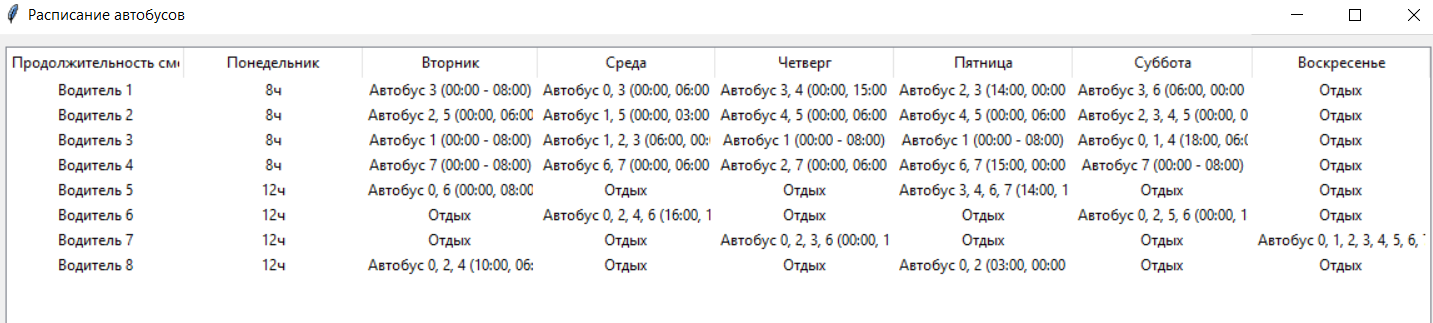


Рис 15 – Вывод

Генетический Алгоритм

То же самое, только теперь к этому условию будем применять генетический алгоритм

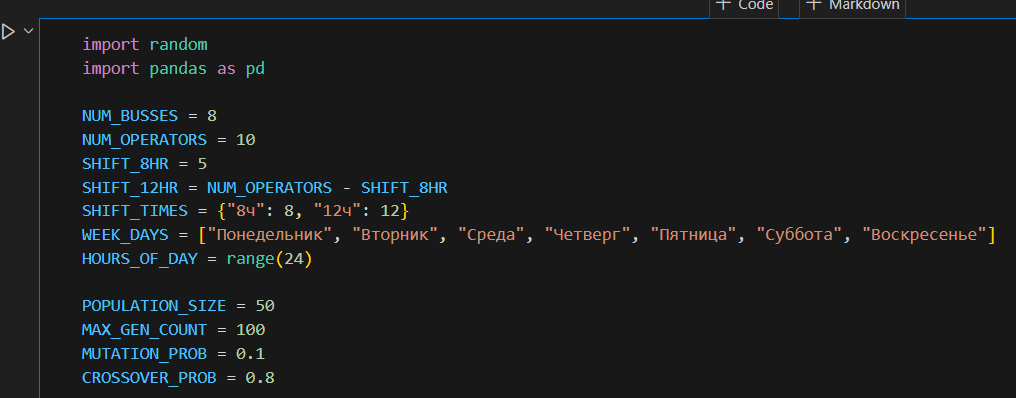


Рис 16 – Параметры и константы

Эти параметры и константы описывают размеры задачи. POPULATION\_SIZE и другие параметры генетического алгоритма настраивают его поведение

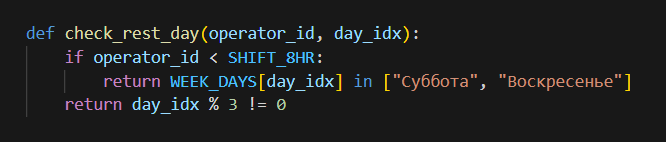


Рис 17 – Проверка на выходно день

Эта функция проверяет, есть ли у водителя выходной в данный день. Если водитель работает на 8-часовой смене, он отдыхает в субботу или воскресенье. Для водителей на 12-часовой смене выходной через 3 дня работы.

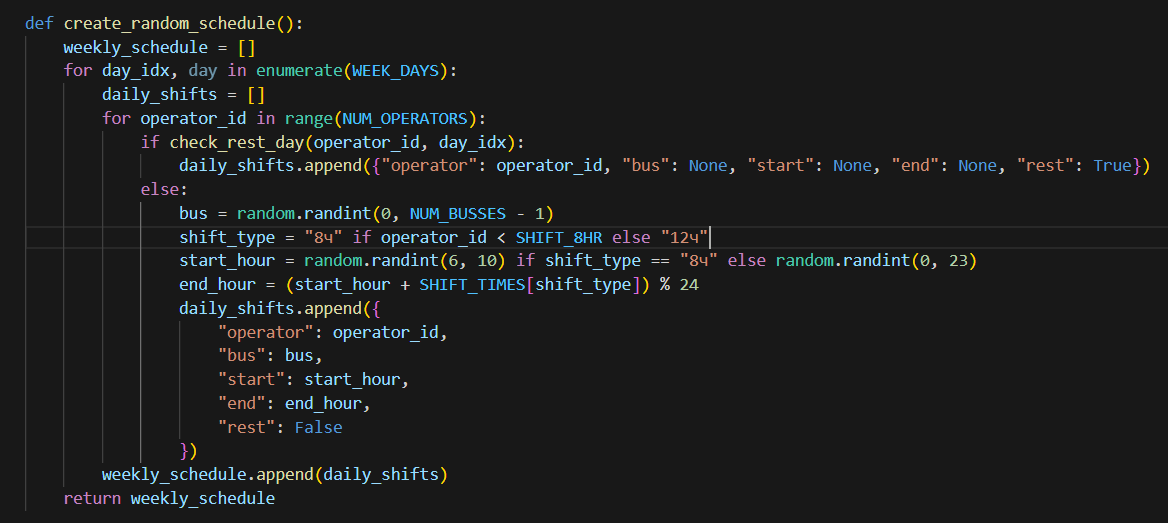


Рис 18 – Генерация расписания

Эта функция генерирует случайное расписание для всех водителей и автобусов на неделю. Она проверяет, может ли водитель отдыхать в данный день, и если не отдыхает, то назначает смену с автобусом, временем начала и окончания. Каждая смена длится либо 8, либо 12 часов.

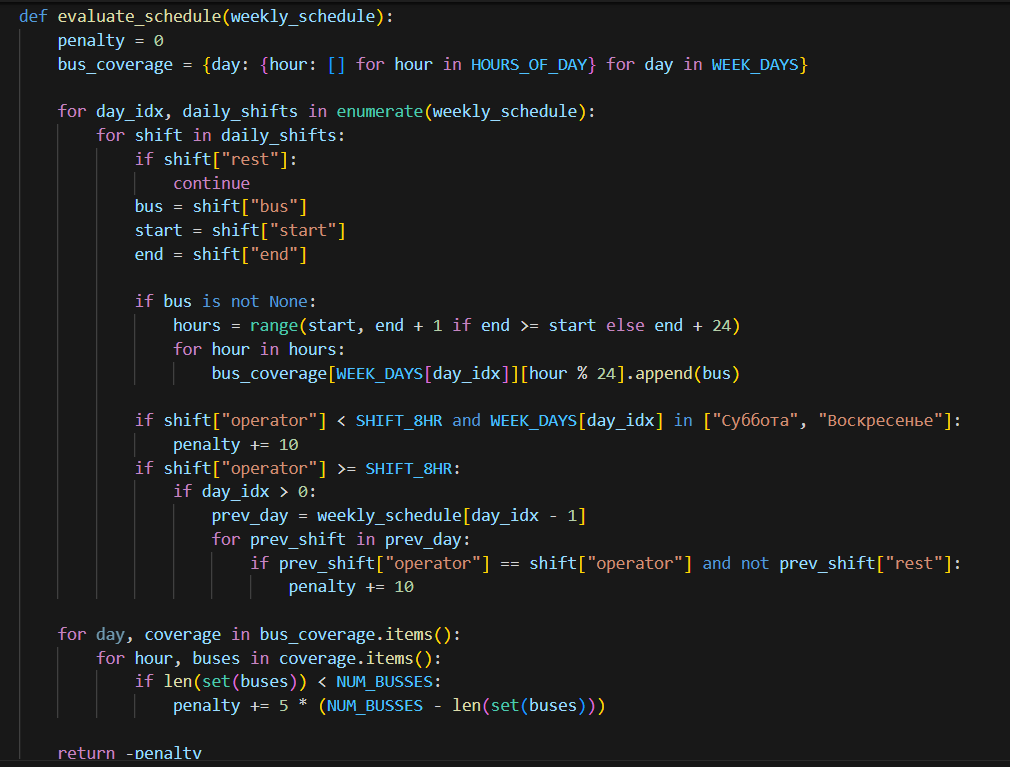


Рис 19 – Оценка расписания, учитывая штрафы

Суть такова:

1. Если водители на 8-часовой смене работают в выходные, начисляется штраф.
2. Если один и тот же водитель работает два дня подряд без отдыха, также начисляется штраф.
3. Если автобус не покрывает все часы дня, начисляется штраф за недостаток автобусов.

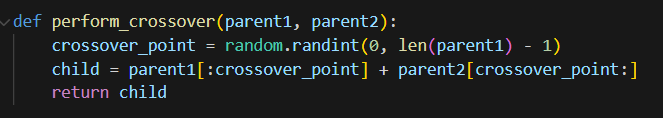


Рис 20 – Объединение

Здесь происходит объединение двух родителей, где точка пересечения случайным образом выбирается в расписании

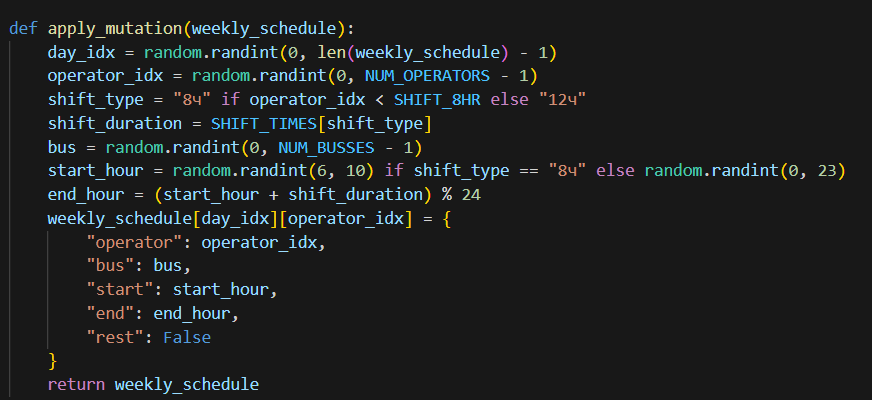


Рис 21 – Мутация

Мутация заключается в случайном изменении одного из водителей в расписании — меняется его смена (автобус, время начала и окончания).

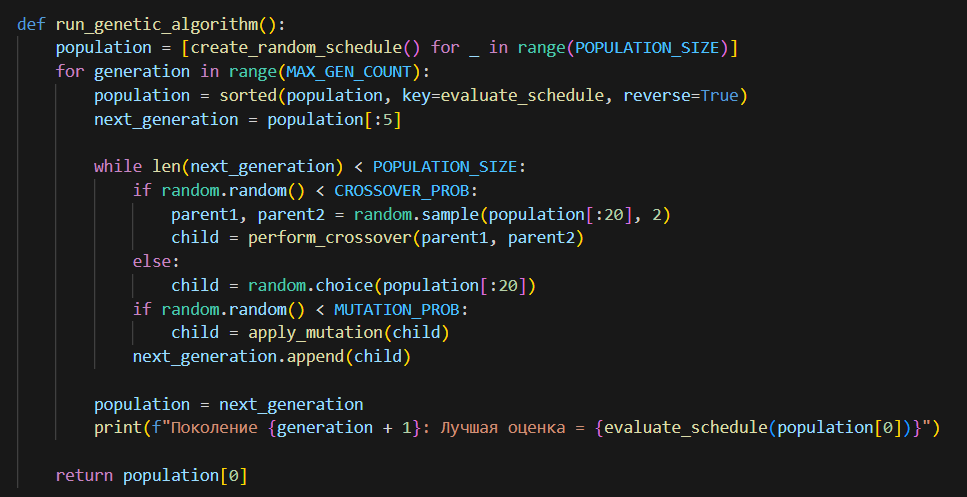


Рис 22 –Генетичесий алгоритм

Этот алгоритм создаёт популяцию случайных расписаний и в каждом поколении улучшает их с помощью объединения и мутации. После каждого поколения выбирается "лучшая" особь, которая будет продолжать участвовать в следующем поколении.

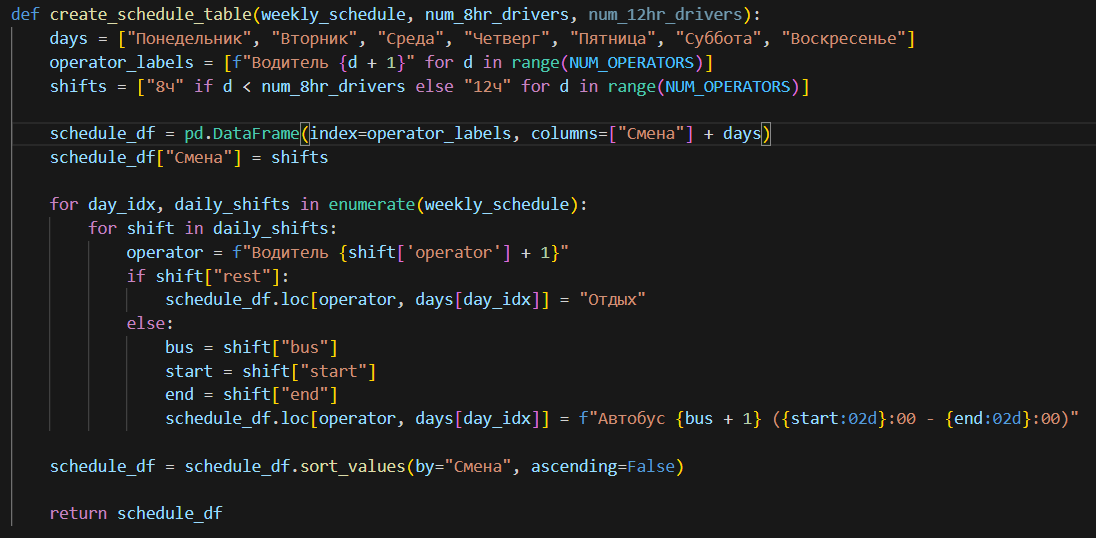


Рис 23 – Таблицы с расписанием

Эта функция создаёт таблицу с расписанием для отображения. Каждый водитель распределён по дням недели, и для каждого дня указан автобус, время начала и окончания смены или "Отдых", если водитель не работает.



Рис 24 – Стиль для таблицы

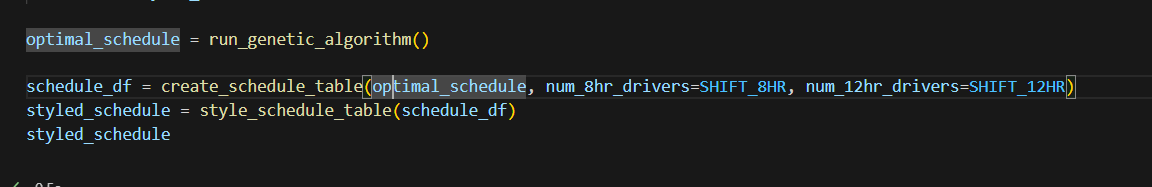


Рис 25 – Запуск и вывод

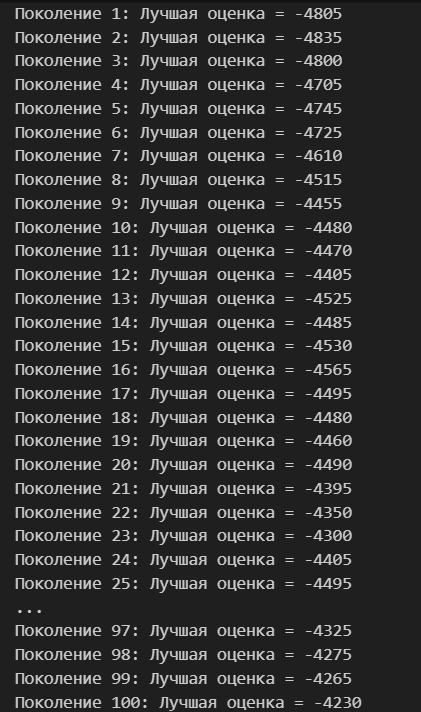
 

Рис 25 - Результаты

**Сравнение алгоритмов**

**В лоб**

1. Он всегда гарантирует нахождение точного оптимального решения за счет перебора всех варинатов
2. У него плохая масштабируемость, так как может расти размер задачи
3. У него простая реализация, но при этом он она становится невозможной для огромных задач
4. Всегда выдает одинаковый результат
5. 100% точность

**Генетический алгоритм**

1. Предоставляет приближенные решения
2. Хорошая масштабируемость, хорошо справляется с большими задачами
3. Сложная реализация из-за настройки параметров
4. Результаты могут отличаться из-за случайности в выборе мутаций и др. параметров
5. Результат высокий, но иногда может упускать 100% точноть

**Итог**

Итогом работы стало создание алгоритмов для автоматического составления расписания водителей автобусов с учетом смен, выходных и нагрузки. Полученные решения позволяют оптимально распределить работу водителей и визуализировать расписание в таблице.

Список источников

1. В. Хилльер, Дж. Либерман. "Введение в математическое программирование и оптимизацию".
2. Д. Бертсек и Дж. Царацис. "Линейное и нелинейное программирование" (Linear and Nonlinear Programming).
3. Mitchell, M. (1998). An Introduction to Genetic Algorithms.
4. Официальный сайт Tkinter: <https://docs.python.org/3/library/tkinter.html>
5. Python Documentation: <https://docs.python.org/3/>.
6. Видеоролики про оптимизацию на Python в ютуб

**Листинг кода**

**Brute\_Force.ipynb**

import pulp

from pulp import LpMinimize, LpProblem, LpVariable, lpSum, LpBinary, LpStatus

from pulp.apis import PULP\_CBC\_CMD

import pandas as pd

import tkinter as tk

from tkinter import ttk

num\_bus = 8

days\_of\_schedule = range(7)

total\_drivers = 8

num\_short\_shift\_drivers = 4

num\_long\_shift\_drivers = total\_drivers - num\_short\_shift\_drivers

hours\_in\_day = range(24)

problem = LpProblem(name="Составление расписания", sense=LpMinimize)

assign = LpVariable.dicts(

    "назначить", [(drv, coach, day, hour) for drv in range(total\_drivers)

                for coach in range(num\_bus) for day in days\_of\_schedule for hour in hours\_in\_day],

    cat=LpBinary

)

for bus in range(num\_bus):

    for day in days\_of\_schedule:

        for hour in hours\_in\_day:

            problem += lpSum(assign[drv, bus, day, hour] for drv in range(total\_drivers)) <= 1

for drv in range(total\_drivers):

    for day in days\_of\_schedule:

        shift\_duration = 8 if drv < num\_short\_shift\_drivers else 12

        for shift\_start in range(0, 24 - shift\_duration + 1):

            for bus in range(num\_bus):

                if any(assign[drv, bus, day, hour].varValue == 1 for hour in range(shift\_start, shift\_start + shift\_duration) if hour < 24):

                    for hour in range(shift\_start, shift\_start + shift\_duration):

                        if hour < 24:

                            problem += assign[drv, bus, day, hour] == 1

for drv in range(total\_drivers):

    for day in days\_of\_schedule:

        for bus in range(num\_bus):

            for hour in hours\_in\_day[:-1]:

                problem += assign[drv, bus, day, hour + 1] >= assign[drv, bus, day, hour]

for bus in range(num\_bus):

    for day in days\_of\_schedule:

        for hour in hours\_in\_day:

            problem += lpSum(assign[drv, bus, day, hour] for drv in range(total\_drivers)) == 1

for bus in range(num\_bus):

    for day in days\_of\_schedule:

        for hour in hours\_in\_day:

            problem += (

                lpSum(assign[drv, bus, day, hour] for drv in range(num\_short\_shift\_drivers)) +

                lpSum(assign[drv, bus, day, hour] for drv in range(num\_short\_shift\_drivers, total\_drivers))

                <= 1

            )

for drv in range(num\_short\_shift\_drivers):

    for day in days\_of\_schedule:

        if day < 5:

            for bus in range(num\_bus):

                for hour in range(6, 14):

                    problem += assign[drv, bus, day, hour] == assign[drv, bus, 0, 6]

        else:

            problem += lpSum(assign[drv, coach, day, hour] for coach in range(num\_bus) for hour in hours\_in\_day) == 0

for drv in range(num\_short\_shift\_drivers, total\_drivers):

    start\_day = (drv - num\_short\_shift\_drivers) % 3

    for day in days\_of\_schedule:

        if ((day - start\_day) % 3 + 3) % 3 == 0:

            for bus in range(num\_bus):

                problem += lpSum(assign[drv, bus, day, hour] for hour in range(14, 24)) == 10

                if day < 6:

                    problem += lpSum(assign[drv, bus, day + 1, hour] for hour in range(0, 12)) == 12

        else:

            problem += lpSum(assign[drv, coach, day, hour] for coach in range(num\_bus) for hour in hours\_in\_day) == 0

for drv in range(total\_drivers):

    for day in days\_of\_schedule:

        for hour in hours\_in\_day:

            problem += lpSum(assign[drv, coach, day, hour] for coach in range(num\_bus)) <= 1

for drv in range(num\_short\_shift\_drivers):

    problem += lpSum(assign[drv, coach, 0, 6] for coach in range(num\_bus)) == 1

for bus in range(num\_bus):

    for hour in range(6, 24):

        problem += lpSum(assign[drv, bus, 0, hour] for drv in range(total\_drivers)) == 1

problem += lpSum(assign[drv, coach, day, hour] for drv in range(total\_drivers)

                 for coach in range(num\_bus) for day in days\_of\_schedule for hour in hours\_in\_day)

status = problem.solve(pulp.PULP\_CBC\_CMD(msg=1))

print(f"Статус: {LpStatus[status]}")

shift\_start\_times = {"8ч": "06:00", "12ч": "06:00"}

shift\_durations = {"8ч": 8, "12ч": 12}

days = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье"]

driver\_labels = [f"Водитель {drv + 1}" for drv in range(total\_drivers)]

schedule\_table = pd.DataFrame(index=driver\_labels, columns=days)

for drv in range(total\_drivers):

    shift\_type = "8ч" if drv < num\_short\_shift\_drivers else "12ч"

    duration = shift\_durations[shift\_type]

    for day in days\_of\_schedule:

        assigned\_coaches = []

        start\_times = []

        end\_times = []

        for bus in range(num\_bus):

            assigned\_hours = [hour for hour in hours\_in\_day if assign[drv, bus, day, hour].varValue == 1]

            if assigned\_hours:

                assigned\_coaches.append(bus)

                start\_time = min(assigned\_hours)

                end\_time = (start\_time + duration) % 24

                start\_times.append(f"{start\_time:02d}:00")

                end\_times.append(f"{end\_time:02d}:00")

        if assigned\_coaches:

            coach\_assignments = ", ".join(map(str, assigned\_coaches))

            shift\_start = ", ".join(start\_times)

            shift\_end = ", ".join(end\_times)

            schedule\_table.iloc[drv, day] = f"Автобус {coach\_assignments} ({shift\_start} - {shift\_end})"

        else:

            schedule\_table.iloc[drv, day] = "Отдых"

shift\_types = ["8ч" if drv < num\_short\_shift\_drivers else "12ч" for drv in range(total\_drivers)]

schedule\_table.insert(0, "Продолжительность смены", shift\_types)

def show\_table\_in\_new\_window(dataframe):

    window = tk.Tk()

    window.title("Расписание автобусов")

    window.geometry("1000x500")

    tree = ttk.Treeview(window, columns=list(dataframe.columns), show="headings", height=len(dataframe))

    for col in dataframe.columns:

        tree.heading(col, text=col, anchor=tk.CENTER)

        tree.column(col, anchor=tk.CENTER, width=130)

    for index, row in dataframe.iterrows():

        tree.insert("", "end", values=[index] + list(row))

    tree.pack(expand=True, fill="both", padx=10, pady=10)

    close\_button = tk.Button(window, text="Закрыть", command=window.destroy, bg="#d9534f", fg="white", padx=10, pady=5)

    close\_button.pack(pady=10)

    window.mainloop()

show\_table\_in\_new\_window(schedule\_table)

**Gen\_algorythm\_buses.ipynb**

import random

import pandas as pd

NUM\_BUSSES = 8

NUM\_OPERATORS = 10

SHIFT\_8HR = 5

SHIFT\_12HR = NUM\_OPERATORS - SHIFT\_8HR

SHIFT\_TIMES = {"8ч": 8, "12ч": 12}

WEEK\_DAYS = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье"]

HOURS\_OF\_DAY = range(24)

POPULATION\_SIZE = 50

MAX\_GEN\_COUNT = 100

MUTATION\_PROB = 0.1

CROSSOVER\_PROB = 0.8

def check\_rest\_day(operator\_id, day\_idx):

    if operator\_id < SHIFT\_8HR:

        return WEEK\_DAYS[day\_idx] in ["Суббота", "Воскресенье"]

    return day\_idx % 3 != 0

def create\_random\_schedule():

    weekly\_schedule = []

    for day\_idx, day in enumerate(WEEK\_DAYS):

        daily\_shifts = []

        for operator\_id in range(NUM\_OPERATORS):

            if check\_rest\_day(operator\_id, day\_idx):

                daily\_shifts.append({"operator": operator\_id, "bus": None, "start": None, "end": None, "rest": True})

            else:

                bus = random.randint(0, NUM\_BUSSES - 1)

                shift\_type = "8ч" if operator\_id < SHIFT\_8HR else "12ч"

                start\_hour = random.randint(6, 10) if shift\_type == "8ч" else random.randint(0, 23)

                end\_hour = (start\_hour + SHIFT\_TIMES[shift\_type]) % 24

                daily\_shifts.append({

                    "operator": operator\_id,

                    "bus": bus,

                    "start": start\_hour,

                    "end": end\_hour,

                    "rest": False

                })

        weekly\_schedule.append(daily\_shifts)

    return weekly\_schedule

def evaluate\_schedule(weekly\_schedule):

    penalty = 0

    bus\_coverage = {day: {hour: [] for hour in HOURS\_OF\_DAY} for day in WEEK\_DAYS}

    for day\_idx, daily\_shifts in enumerate(weekly\_schedule):

        for shift in daily\_shifts:

            if shift["rest"]:

                continue

            bus = shift["bus"]

            start = shift["start"]

            end = shift["end"]

            if bus is not None:

                hours = range(start, end + 1 if end >= start else end + 24)

                for hour in hours:

                    bus\_coverage[WEEK\_DAYS[day\_idx]][hour % 24].append(bus)

            if shift["operator"] < SHIFT\_8HR and WEEK\_DAYS[day\_idx] in ["Суббота", "Воскресенье"]:

                penalty += 10

            if shift["operator"] >= SHIFT\_8HR:

                if day\_idx > 0:

                    prev\_day = weekly\_schedule[day\_idx - 1]

                    for prev\_shift in prev\_day:

                        if prev\_shift["operator"] == shift["operator"] and not prev\_shift["rest"]:

                            penalty += 10

    for day, coverage in bus\_coverage.items():

        for hour, buses in coverage.items():

            if len(set(buses)) < NUM\_BUSSES:

                penalty += 5 \* (NUM\_BUSSES - len(set(buses)))

    return -penalty

def perform\_crossover(parent1, parent2):

    crossover\_point = random.randint(0, len(parent1) - 1)

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

    return child

def apply\_mutation(weekly\_schedule):

    day\_idx = random.randint(0, len(weekly\_schedule) - 1)

    operator\_idx = random.randint(0, NUM\_OPERATORS - 1)

    shift\_type = "8ч" if operator\_idx < SHIFT\_8HR else "12ч"

    shift\_duration = SHIFT\_TIMES[shift\_type]

    bus = random.randint(0, NUM\_BUSSES - 1)

    start\_hour = random.randint(6, 10) if shift\_type == "8ч" else random.randint(0, 23)

    end\_hour = (start\_hour + shift\_duration) % 24

    weekly\_schedule[day\_idx][operator\_idx] = {

        "operator": operator\_idx,

        "bus": bus,

        "start": start\_hour,

        "end": end\_hour,

        "rest": False

    }

    return weekly\_schedule

def run\_genetic\_algorithm():

    population = [create\_random\_schedule() for \_ in range(POPULATION\_SIZE)]

    for generation in range(MAX\_GEN\_COUNT):

        population = sorted(population, key=evaluate\_schedule, reverse=True)

        next\_generation = population[:5]

        while len(next\_generation) < POPULATION\_SIZE:

            if random.random() < CROSSOVER\_PROB:

                parent1, parent2 = random.sample(population[:20], 2)

                child = perform\_crossover(parent1, parent2)

            else:

                child = random.choice(population[:20])

            if random.random() < MUTATION\_PROB:

                child = apply\_mutation(child)

            next\_generation.append(child)

        population = next\_generation

        print(f"Поколение {generation + 1}: Лучшая оценка = {evaluate\_schedule(population[0])}")

    return population[0]

def create\_schedule\_table(weekly\_schedule, num\_8hr\_drivers, num\_12hr\_drivers):

    days = ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье"]

    operator\_labels = [f"Водитель {d + 1}" for d in range(NUM\_OPERATORS)]

    shifts = ["8ч" if d < num\_8hr\_drivers else "12ч" for d in range(NUM\_OPERATORS)]

    schedule\_df = pd.DataFrame(index=operator\_labels, columns=["Смена"] + days)

    schedule\_df["Смена"] = shifts

    for day\_idx, daily\_shifts in enumerate(weekly\_schedule):

        for shift in daily\_shifts:

            operator = f"Водитель {shift['operator'] + 1}"

            if shift["rest"]:

                schedule\_df.loc[operator, days[day\_idx]] = "Отдых"

            else:

                bus = shift["bus"]

                start = shift["start"]

                end = shift["end"]

                schedule\_df.loc[operator, days[day\_idx]] = f"Автобус {bus + 1} ({start:02d}:00 - {end:02d}:00)"

    schedule\_df = schedule\_df.sort\_values(by="Смена", ascending=False)

    return schedule\_df

def style\_schedule\_table(schedule\_df):

    styled\_schedule = (

        schedule\_df.style.set\_properties(

            \*\*{

                "text-align": "center",

                "font-size": "14px",

                "border": "1px solid #ddd",

                "padding": "8px",

                "background-color": "#f5f5f5",

            }

        )

        .set\_table\_styles(

            [

                {"selector": "th", "props": [("background-color", "#4CAF50"), ("color", "white"), ("font-size", "16px"), ("font-weight", "bold")]},

                {"selector": "td", "props": [("background-color", "#ffffff"), ("font-size", "14px"), ("font-weight", "normal"), ("color", "#333333")]},

                {"selector": "tr:nth-child(odd)", "props": [("background-color", "#f9f9f9")]},

                {"selector": "tr:nth-child(even)", "props": [("background-color", "#ffffff")]},

                {"selector": "td, th", "props": [("border", "1px solid #ddd")]},

                {"selector": "th", "props": [("padding", "10px")]},

                {"selector": "td", "props": [("padding", "10px")]},

            ]

        )

    )

    return styled\_schedule

optimal\_schedule = run\_genetic\_algorithm()

schedule\_df = create\_schedule\_table(optimal\_schedule, num\_8hr\_drivers=SHIFT\_8HR, num\_12hr\_drivers=SHIFT\_12HR)

styled\_schedule = style\_schedule\_table(schedule\_df)

styled\_schedule

**Github:** https://github.com/NordBeard/Kursovaya\_Rabota\_SYAOD.git