**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»

(МТУСИ)

Кафедра «МКиИТ»

Отчет по курсовой работе

По дисциплине: «Структуры и алгоритмы обработки данных»

На тему: «Оптимизация расписания водителей»

|  |  |
| --- | --- |
| Выполнил: | студент группы  БВТ2205 Рыбкин А.А. |
| Проверил: | **Симонов С. Е.** |

**Аннотация**

Данный проект посвящен разработке программного обеспечения для оптимизации расписания водителей с использованием генетического алгоритма. Программа позволяет эффективно распределять время работы водителей, минимизируя время простоя и учитывая различные ограничения, такие как продолжительность смен и минимальные интервалы между маршрутами. В результате работы была создана программа с графическим интерфейсом, которая позволяет пользователям вводить параметры и получать оптимизированное расписание.

Оглавление

[1. Введение 4](#_Toc185183428)

[2. Основная часть 5](#_Toc185183429)

[1. Цели и задачи проекта 5](#_Toc185183430)

[3. Методология 6](#_Toc185183431)

[3.1. Генетический алгоритм 6](#_Toc185183432)

[3.2. Валидация расписания 6](#_Toc185183433)

[4. Реализация 6](#_Toc185183434)

[4.1. Структура программы 6](#_Toc185183435)

[4.2. Графический интерфейс 7](#_Toc185183436)

[5. Заключение 7](#_Toc185183437)

[6. Список использованных источников 8](#_Toc185183438)

[7. Приложения 8](#_Toc185183439)

[7.1. genetic\_algorithm.py: 8](#_Toc185183440)

[7.2. gui.py: 10](#_Toc185183441)

[7.3. main.py 12](#_Toc185183442)

[7.4. route\_generation: 12](#_Toc185183443)

[7.5. schedule\_validation.py: 13](#_Toc185183444)

# Введение

Актуальность разработки программного обеспечения для оптимизации расписания водителей обусловлена необходимостью повышения эффективности работы транспортных компаний. Существующие методы планирования часто не учитывают все ограничения и могут приводить к неэффективному использованию ресурсов. Генетические алгоритмы, как один из методов оптимизации, позволяют находить решения в сложных задачах, что делает их подходящими для данной области.

# Основная часть

## Цели и задачи проекта

Целью проекта является создание программного обеспечения, которое позволит оптимизировать расписание водителей. Для достижения этой цели были поставлены следующие задачи:

* Исследование методов оптимизации расписаний.
* Разработка алгоритма для генерации оптимального расписания.
* Создание графического интерфейса для удобного взаимодействия с пользователем.
* Валидация сгенерированного расписания на соответствие заданным ограничениям.
* Тестирование и отладка программы.

# Методология

## 3.1. Генетический алгоритм

Генетический алгоритм является методом оптимизации, который имитирует процесс естественного отбора. Он включает в себя следующие этапы:

Инициализация популяции случайных решений.

Оценка качества решений с помощью функции приспособленности.

Отбор лучших решений для создания нового поколения.

Применение операций скрещивания и мутации для генерации новых решений.

## 3.2. Валидация расписания

Валидация расписания включает в себя проверку на соответствие заданным ограничениям, таким как:

Максимальная продолжительность смены для водителей.

Минимальные интервалы между маршрутами для разных типов водителей.

# Р**еализация**

## 4.1. Структура программы

Программа состоит из нескольких модулей:

main.py: основной файл, запускающий приложение.

gui.py: модуль, отвечающий за создание графического интерфейса.

genetic\_algorithm.py: реализация генетического алгоритма.

route\_generation.py: генерация временных интервалов для маршрутов.

schedule\_validation.py: проверка корректности расписания.

## 4.2. Графический интерфейс

Графический интерфейс разработан с использованием библиотеки Tkinter. Он позволяет пользователю вводить параметры, такие как количество маршрутов и тип водителей, а также отображает сгенерированное расписание в удобном формате.

# Заключение

Разработанное программное обеспечение позволяет эффективно оптимизировать расписание водителей, учитывая различные ограничения и требования. Использование генетического алгоритма обеспечивает высокую степень адаптивности и возможность нахождения оптимальных решений в условиях неопределенности. Программа может быть полезна для транспортных компаний, стремящихся повысить эффективность работы своих водителей.

# Список использованных источников

1. Баранов, Н. А. Генетические алгоритмы в решении задач оптимизации. М.: Наука, 2018.
2. Бульдиков, В. И. Оптимизация процессов. СПб.: Питер, 2020.
3. Жданов, А. М. Применение генетических алгоритмов для распределения ресурсов. М.: Инфра-М, 2019.
4. Исаев, И. В. Современные методы оптимизации. М.: Финансы и статистика, 2021.
5. Кузнецов, С. А. Генетические алгоритмы: теория и практика. М.: Высшая школа, 2019.
6. Левин, П. Р. Алгоритмы и структуры данных. М.: БХВ-Петербург, 2020.
7. Михайлов, В. В. Операционные исследования. М.: Эксмо, 2019.

# Приложения

## 7.1. genetic\_algorithm.py:

1. import random
2. from datetime import datetime, timedelta
3. from schedule\_validation import validate\_schedule, fitness
4. def genetic\_schedule(driver\_list, driver\_type, route\_times, num\_routes, traffic\_route\_time, shift\_duration\_A, population\_size=100, max\_generations=50):
6. # Инициализация популяции случайных расписаний
7. population = [{driver: [] for driver in driver\_list} for \_ in range(population\_size)]
9. # Заполнение каждого расписания случайными маршрутами
10. for schedule in population:
11. for \_ in range(num\_routes):
12. driver = random.choice(driver\_list)  # Случайный выбор водителя
13. start\_time = random.choice(route\_times)  # Случайный выбор времени начала маршрута
14. end\_time = calculate\_route\_end(start\_time, traffic\_route\_time)  # Вычисление времени окончания маршрута
15. schedule[driver].append((start\_time, end\_time))  # Добавление маршрута в расписание
16. # Основной цикл генетического алгоритма
17. for \_ in range(max\_generations):
18. # Сортировка популяции по функции приспособленности
19. population.sort(key=lambda x: fitness(x, driver\_type, shift\_duration\_A), reverse=True)
21. # Сохранение лучших 10 расписаний для следующего поколения
22. next\_population = population[:10]
24. # Генерация новых расписаний до достижения необходимого размера популяции
25. while len(next\_population) < population\_size:
26. parent1, parent2 = random.sample(population[:50], 2)  # Случайный выбор двух родителей из лучших 50
27. # Создание потомка, выбирая маршруты от обоих родителей
28. child = {driver: random.choice([parent1[driver], parent2[driver]]) for driver in driver\_list}
30. # С вероятностью 20% выполняется мутация
31. if random.random() < 0.2:
32. driver = random.choice(driver\_list)  # Случайный выбор водителя для мутации
33. if child[driver]:  # Если у водителя есть маршруты
34. child[driver].pop(random.randint(0, len(child[driver]) - 1))  # Удаление случайного маршрута
35. start\_time = random.choice(route\_times)  # Случайный выбор нового времени начала маршрута
36. end\_time = calculate\_route\_end(start\_time, traffic\_route\_time)  # Вычисление времени окончания
37. child[driver].append((start\_time, end\_time))  # Добавление нового маршрута
39. next\_population.append(child)  # Добавление потомка в следующее поколение
41. population = next\_population  # Обновление популяции
42. # Возвращение лучшего расписания на основе функции приспособленности
43. return max(population, key=lambda x: fitness(x, driver\_type, shift\_duration\_A))
44. def calculate\_route\_end(start\_time, route\_duration):
45. return (datetime.combine(datetime.today(), start\_time) + route\_duration).time()  # Возвращает время окончания

## 7.2. gui.py:

import tkinter as tk

from tkinter import ttk

from genetic\_algorithm import genetic\_schedule

from route\_generation import generate\_route\_times

from datetime import timedelta

# Определение глобальных переменных

peak\_hours = [(7, 9), (17, 19)]

non\_peak\_hours = [(6, 7), (9, 17), (19, 3)]

traffic\_route\_time = timedelta(minutes=70)

drivers\_A = ["Driver\_A1", "Driver\_A2", "Driver\_A3"]

drivers\_B = ["Driver\_B1", "Driver\_B2"]

shift\_duration\_A = timedelta(hours=8)

def generate\_schedule(driver\_type, drivers, num\_routes, route\_times, tree):

    try:

        if not drivers:

            for i in tree.get\_children():

                tree.delete(i)

            tree.insert("", "end", values=(f"Нет водителей типа {driver\_type}.", "", ""))

            return

        best\_schedule = genetic\_schedule(drivers, driver\_type, route\_times, num\_routes, traffic\_route\_time, shift\_duration\_A)

        display\_schedule(best\_schedule, tree)

    except ValueError:

        for i in tree.get\_children():

            tree.delete(i)

        tree.insert("", "end", values=("Ошибка: Введите корректные параметры.", "", ""))

def display\_schedule(schedule, tree):

    for i in tree.get\_children():

        tree.delete(i)  # Очистка существующих строк

    for driver, routes in schedule.items():

        for start, end in routes:

            tree.insert("", "end", values=(driver, start.strftime('%H:%M'), end.strftime('%H:%M')))

def create\_gui():

    root = tk.Tk()

    root.title("Оптимальное расписание")

    root.geometry("600x400")

    root.configure(bg="#2E2E2E")

    # Конфигурация стиля

    style = ttk.Style()

    style.configure("TButton", background="#4CAF50", foreground="black", font=("Helvetica", 12))

    style.configure("TEntry", font=("Helvetica", 12))

    style.configure("Treeview", font=("Helvetica", 12), background="#FFFFFF", foreground="#000000")

    style.configure("Treeview.Heading", font=("Helvetica", 12, 'bold'))

    # Метка для ввода количества маршрутов

    num\_routes\_label = tk.Label(root, text="Количество маршрутов:", bg="#2E2E2E", fg="red", font=("Helvetica", 12))

    num\_routes\_label.pack(pady=5)

    num\_routes\_entry = ttk.Entry(root, width=10)

    num\_routes\_entry.insert(0, "10")

    num\_routes\_entry.pack(pady=5)

    # Создание Treeview для отображения расписания

    columns = ("Driver", "Start Time", "End Time")

    tree = ttk.Treeview(root, columns=columns, show='headings')

    tree.heading("Driver", text="Водитель")

    tree.heading("Start Time", text="Время начала")

    tree.heading("End Time", text="Время окончания")

    tree.pack(pady=10, fill=tk.BOTH, expand=True)

    # Генерация времени маршрутов

    route\_times = generate\_route\_times(peak\_hours, non\_peak\_hours, traffic\_route\_time)

    # Информационная метка

    info\_label = tk.Label(root, text="Нажмите кнопку для генерации расписания для выбранного типа водителей.", bg="#2E2E2E", fg="white", font=("Helvetica", 10))

    info\_label.pack(pady=5)

    # Кнопки для генерации расписаний

    button\_frame = tk.Frame(root, bg="#2E2E2E")

    button\_frame.pack(pady=5)

    type\_a\_button = ttk.Button(button\_frame, text="Генерировать для Тип A", command=lambda: generate\_schedule("A", drivers\_A, int(num\_routes\_entry.get()), route\_times, tree))

    type\_a\_button.pack(side=tk.LEFT, padx=10)

    type\_b\_button = ttk.Button(button\_frame, text="Генерировать для Тип B", command=lambda: generate\_schedule("B", drivers\_B, int(num\_routes\_entry.get()), route\_times, tree))

    type\_b\_button.pack(side=tk.LEFT, padx=10)

    # Кнопка выхода

    exit\_button = ttk.Button(root, text="Выход", command=root.quit)

    exit\_button.pack(pady=10)

    root.mainloop()

7.3. main.py:

from gui import create\_gui

if \_\_name\_\_ == "\_\_main\_\_":

    create\_gui()

## 7.4. route\_generation:

from datetime import datetime, timedelta

def generate\_route\_times(peak\_hours, non\_peak\_hours, traffic\_route\_time):

    route\_times = []  # Инициализация списка для хранения временных интервалов

    # Объединение пиковых и непиковых часов для обработки

    for hours in peak\_hours + non\_peak\_hours:

        # Преобразование времени начала маршрута в объект datetime

        current\_time = datetime.strptime(f"{hours[0]}:00", "%H:%M")

        # Цикл для генерации временных интервалов до конца указанного часа

        while current\_time.hour != hours[1]:

            route\_times.append(current\_time.time())  # Добавление текущего времени в список

            current\_time += traffic\_route\_time  # Увеличение текущего времени на время маршрута

    return route\_times  # Возвращение списка временных интервалов

## 7.5. schedule\_validation.py:

from datetime import datetime, timedelta

def is\_time\_overlap(start1, end1, start2, end2):

    return max(start1, start2) < min(end1, end2)  # Проверка на пересечение интервалов

def validate\_schedule(schedule, driver\_type, shift\_duration\_A):

    """

    Проверяет корректность расписания для водителей.

    """

    for routes in schedule.values():  # Проходим по маршрутам каждого водителя

        if driver\_type == "A":

            total\_time = timedelta()  # Инициализируем общее время как timedelta

            for start, end in routes:

                # Преобразуем время начала и окончания в объекты datetime

                start\_time = datetime.combine(datetime.today(), start)

                end\_time = datetime.combine(datetime.today(), end)

                total\_time += (end\_time - start\_time)  # Суммируем продолжительность маршрутов

            # Проверяем, не превышает ли общее время допустимое значение

            if total\_time > (shift\_duration\_A - timedelta(hours=1)):

                return False  # Если превышает, возвращаем False

        elif driver\_type == "B":

            # Проверяем, что между маршрутами водителей типа B есть минимум 15 минут

            for i in range(1, len(routes)):

                start1 = datetime.combine(datetime.today(), routes[i-1][1])  # Время окончания предыдущего маршрута

                start2 = datetime.combine(datetime.today(), routes[i][0])  # Время начала текущего маршрута

                if (start2 - start1) < timedelta(minutes=15):

                    return False  # Если разница меньше 15 минут, возвращаем False

    return True  # Если все проверки пройдены, возвращаем True

def fitness(schedule, driver\_type, shift\_duration\_A):

    """

    Вычисляет функцию приспособленности для оценки качества расписания.

    """

    # Подсчет штрафов за пересечение временных интервалов

    penalties = sum(

        is\_time\_overlap(r1[0], r1[1], r2[0], r2[1])

        for routes in schedule.values()

        for i, r1 in enumerate(routes)

        for r2 in routes[i+1:]  # Сравниваем каждый маршрут с последующими

    )

    # Проверка корректности расписания и добавление штрафа, если оно некорректно

    if not validate\_schedule(schedule, driver\_type, shift\_duration\_A):

        penalties += 10  # Добавляем штраф за некорректное расписание

    return -penalties  # Возвращаем отрицательное значение штрафов для минимизации

Ссылка на проект: https://github.com/Arturrybkin/-