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 «Структуры и алгоритмы обработки данных»

КУРСОВАЯ РАБОТА

По дисциплине «Структуры и алгоритмы обработки данных»

Выполнил:
Абделжауед Мохамед Али
Группа: БВТ2204
Проверил:
Симонов

Москва, 2024г.

Оглавление

Цель работы : 3

Задачи : 3

1. Введение : 4

2. Вводные данные и структура проблемы : 5

3. Реализация : 6

3.1 Метод 'влоб : 6

3.2 Генетический алгоритм : 6

3.3 Результаты работы методов : 8

4. Сравнение методов : 9

5. Ограничения и их реализация : 10

5.1. Проверка типа маршрута: 10

5.2. Проверка пиковых часов: 10

5.3. Добавление времени старта маршрута: 10

5.4 Обновленный вывод: 10

5.5 результаты обоих методов : 13

Заключение 14

Список литературы 22

Цель работы :

разработать алгоритмы и программное обеспечение для оптимизации расписания маршрутных автобусов, минимизируя затраты и соблюдая ограничения, такие как рабочее время водителей, перерывы и часы пик.

Задачи :

1. Проанализировать входные данные и ограничения задачи.
2. Реализовать два метода решения: метод "влоб" и генетический алгоритм.
3. Сравнить методы по критериям: оптимальность решения, удобство использования программы, качество документации.
4. Предоставить расписание и выводы в отчёте.

1. Введение :

Оптимизация расписания маршрутных автобусов играет важную роль в эффективном управлении городскими транспортными системами. В данной работе рассматривается задача создания оптимального расписания, которое минимизирует использование ресурсов автопарка и удовлетворяет ограничениям. Для решения задачи используются два подхода: метод полного перебора и генетический алгоритм.

2. Вводные данные и структура проблемы :

Структура проблемы определяется с помощью классов Python для моделирования маршрутов, водителей и автобусов. Класс ScheduleManager управляет всеми ресурсами и выводит текущее расписание

```python
# Define the Route class
class Route:
    def __init__(self, name, stops, is_cyclic, time_required):
        self.name = name
        self.stops = stops
        self.is_cyclic = is_cyclic
        self.time_required = time_required

# Define the Driver class
class Driver:
    def __init__(self, driver_type, max_hours, breaks, shift_schedule):
        self.driver_type = driver_type  # 'A' or 'B'
        self.max_hours = max_hours
        self.breaks = breaks
        self.shift_schedule = shift_schedule

# Define the Bus class
class Bus:
    def __init__(self, id, route=None, driver=None):
        self.id = id
        self.route = route
        self.driver = driver

# Define the ScheduleManager to manage routes, buses, and drivers
class ScheduleManager:
    def __init__(self):
        self.routes = []
        self.buses = []
        self.drivers = []

    def add_route(self, route):
        self.routes.append(route)
```

Рисунок 1- Определение структуры задачи

```python
if __name__ == "__main__":
    # Initialize ScheduleManager
    manager = ScheduleManager()

    # Add routes
    manager.add_route(Route("Route 1", [1, 2, 3, 2, 1], True, 60))
    manager.add_route(Route("Route 2", [1, 2, 3, 4, 5, 1], False, 70))

    # Add buses
    for i in range(1, 9):
        manager.add_bus(Bus(i))

    # Add drivers
    manager.add_driver(Driver("A", 80, [4, 1], "06:00-14:00"))  # Increased max_hours
    manager.add_driver(Driver("B", 120, [2, 15], "06:00-06:00"))

    # Print initial schedule
    manager.print_schedule()
```

```
✓ 0.0s
Bus 1: Route None, Driver None
Bus 2: Route None, Driver None
Bus 3: Route None, Driver None
Bus 4: Route None, Driver None
Bus 5: Route None, Driver None
Bus 6: Route None, Driver None
Bus 7: Route None, Driver None
Bus 8: Route None, Driver None
```

Рисунок 2- Определение структуры задачи с результатами

3. Реализация :

В данной работе реализованы два различных подхода для решения задачи оптимизации расписания маршрутных автобусов. Оба метода учитывают ограничения задачи, такие как время работы водителей, пиковые часы и доступность автобусов. Каждый метод имеет свои особенности и применимость:

3.1 Метод 'влоб :

Метод "влоб" предполагает полный перебор всех возможных комбинаций автобусов и водителей для назначения их маршрутам. Этот метод гарантирует нахождение оптимального решения, так как перебираются все варианты. Однако его производительность может быть низкой при большом количестве входных данных.

```python
def brute_force_schedule(manager):
    best_schedule = None
    min_buses_used = float('inf')

    # Generate all permutations of routes
    for route_perm in permutations(manager.routes):
        buses_used = set()
        drivers_used = set()
        current_schedule = []

        for route in route_perm:
            route_assigned = False
            for bus in manager.buses:
                if bus.id not in buses_used:  # Check if the bus is available
                    for driver in manager.drivers:
                        if driver.max_hours >= route.time_required and driver.driver_type not in drivers_used:  # Check driver constraint
                            # Assign this bus and driver to the route
                            current_schedule.append((bus.id, route.name, driver.driver_type))
                            buses_used.add(bus.id)
                            drivers_used.add(driver.driver_type)
                            route_assigned = True
                            break
                    if route_assigned:
                        break

            if not route_assigned:
                print(f"Warning: No valid assignment for route {route.name}.")

        # Check if this schedule uses fewer buses
        if len(buses_used) < min_buses_used:
            min_buses_used = len(buses_used)
            best_schedule = current_schedule

    if not best_schedule:
        print("No valid schedule was created. Please check constraints or input data.")
        return []
```

Рисунок 2- кода Метод "В лоб"

3.2 Генетический алгоритм :

Этот метод имитирует процесс естественной эволюции, создавая популяцию решений, которые с течением поколений улучшаются с помощью механизмов скрещивания, мутации и отбора. Генетический алгоритм позволяет находить приближенные решения значительно быстрее, чем метод 'влоб', особенно для больших и сложных задач.

```python
import random

def genetic_algorithm_schedule(manager, generations=100, population_size=10):
    # Функция для создания случайного расписания
    def create_random_schedule():
        schedule = []
        buses_used = set()
        drivers_used = set()
        for route in manager.routes:
            available_buses = [bus for bus in manager.buses if bus.id not in buses_used]
            if not available_buses:
                print(f"Warning: No available buses for route {route.name}.")
                break
            bus = random.choice(available_buses)

            available_drivers = [driver for driver in manager.drivers if driver.max_hours >= route.time_required and driver.driver_type not in drivers_used]
            if not available_drivers:
                print(f"Warning: No available drivers for route {route.name}.")
                break
            driver = random.choice(available_drivers)

            schedule.append((bus.id, route.name, driver.driver_type))
            buses_used.add(bus.id)
            drivers_used.add(driver.driver_type)
        return schedule



    # Функция для оценки расписания
    def fitness(schedule):
        return len(set([entry[0] for entry in schedule]))  # Количество уникальных автобусов

    # Функция для выполнения скрещивания (crossover)
    def crossover(parent1, parent2):
        split_point = len(parent1) // 2
        child = parent1[:split_point] + parent2[split_point:]
        return child

    # Функция для выполнения мутации (mutation)
    def mutate(schedule):
        index = random.randint(0, len(schedule) - 1)
        used_buses = {entry[0] for entry in schedule if entry != schedule[index]}
        used_drivers = {entry[2] for entry in schedule if entry != schedule[index]}

        available_buses = [bus for bus in manager.buses if bus.id not in used_buses]
        if not available_buses:
            print(f"Warning: No available buses for mutation on route {schedule[index][1]}.")
            return

        available_drivers = [driver for driver in manager.drivers if driver.driver_type not in used_drivers and driver.max_hours >= manager.routes[index].time_
        if not available_drivers:
            print(f"Warning: No available drivers for mutation on route {schedule[index][1]}.")
            return

        bus = random.choice(available_buses)
        driver = random.choice(available_drivers)
        schedule[index] = (bus.id, schedule[index][1], driver.driver_type)

    # Генерация начальной популяции
    population = [create_random_schedule() for _ in range(population_size)]

    # Эволюция популяции
    for generation in range(generations):
        # Оценка приспособленности (fitness)
        population = sorted(population, key=fitness)

        # Отбор лучших решений
        population = population[:population_size // 2]

        # Создание следующего поколения
```

Рисунок 3- кода Метод Генетический алгоритм

```python
        # Создание следующего поколения
        next_generation = []
        while len(next_generation) < population_size:
            parent1, parent2 = random.sample(population, 2)
            child = crossover(parent1, parent2)
            if random.random() < 0.1:  # Вероятность мутации
                mutate(child)
            next_generation.append(child)

        population = next_generation

    # Возврат лучшего расписания
    best_schedule = min(population, key=fitness)
    return best_schedule
```

Рисунок 4-Продолжение кода Метод Генетический алгоритм

3.3 Результаты работы методов :

Оба метода — "в лоб" и генетический алгоритм — успешно сгенерировали расписания, удовлетворяющие всем условиям задачи. Расписания включают уникальные автобусы и водителей для каждого маршрута

```
Generated Schedule (Brute Force):
Bus 1 assigned to Route 1 with Driver A
Bus 2 assigned to Route 2 with Driver B

Generated Schedule (Genetic Algorithm):
Bus 1 assigned to Route 1 with Driver A
Bus 2 assigned to Route 2 with Driver B
```

Рисунок 5- Результаты работы методов

4. **Сравнение методов :**

Оба метода имеют свои преимущества и ограничения:

Таблица 1 - Сравнение методов

Критерий	**Метод "В лоб"**	**Генетический алгоритм**
Оптимальность	Гарантирует оптимальное решение	Находит приближенное решение
Производительность	Медленный для больших данных	Быстрее для больших данных
Простота	Прост в реализации	Сложнее из-за настройки параметров

5. Ограничения и их реализация :

5.1. Проверка типа маршрута:

В функцию добавлена логика для определения типа маршрута:

- Циклический маршрут (Cyclic**)**: Маршрут, который возвращается в начальную точку по круговой схеме.
- Конечный маршрут (Terminal**)**: Маршрут, который заканчивается в исходной точке, но не повторяет путь циклически.

Для определения типа маршрута используется атрибут is_cyclic класса Route.

5.2. Проверка пиковых часов:

Добавлена функция is_in_peak_hours, которая проверяет, попадает ли время старта маршрута в пиковые часы:

- Утро: **7:00–9:00**
- Вечер: 17:00–19:00

Если маршрут начинается в пиковые часы, это выводится в виде примечания: (PEAK HOUR)**.**

5.3. Добавление времени старта маршрута:

Теперь каждый маршрут получает время старта в формате ЧЧ:ММ. Начальное время начинается с 6:00 и увеличивается на 1 час для каждого следующего маршрута.

5.4 Обновленный вывод:

В результат добавлен вывод следующей информации для каждого маршрута:

1. Номер автобуса.
2. Название маршрута.
3. Тип маршрута: Cyclic или Terminal**.**
4. Назначенный водитель.
5. Время старта маршрута.

```python
from itertools import permutations

# Define peak hours
PEAK_HOURS = [(7, 9), (17, 19)]

def is_in_peak_hours(start_time):
    """Check if a route start time is in peak hours."""
    hour = int(start_time.split(":")[0])
    for peak_start, peak_end in PEAK_HOURS:
        if peak_start <= hour < peak_end:
            return True
    return False

def brute_force_schedule(manager):
    best_schedule = None
    min_buses_used = float('inf')

    # Generate all permutations of routes
    for route_perm in permutations(manager.routes):
        buses_used = set()
        drivers_used = set()
        current_schedule = []
        start_time = 6  # Starting from 6:00 AM

        for route in route_perm:
            route_assigned = False
            for bus in manager.buses:
                if bus.id not in buses_used:
                    for driver in manager.drivers:
                        if driver.max_hours >= route.time_required and driver.driver_type not in drivers_used:
                            start_hour = f"{start_time}:00"
                            route_type = "Cyclic" if route.is_cyclic else "Terminal"
                            peak info = " (PEAK HOUR)" if is in peak hours(start hour) else ""

                            print(f"Route {route.name} ({route_type}) starts at {start_hour}{peak_info}")

                            # Assign this bus and driver
                            current_schedule.append((bus.id, route.name, driver.driver_type, start_hour, route_type))
                            buses_used.add(bus.id)
                            drivers_used.add(driver.driver_type)
                            route_assigned = True
                            start_time += 1  # Increment start time
                            break
                    if route_assigned:
                        break

            if not route_assigned:
                print(f"Warning: No valid assignment for route {route.name}.")

        # Check if this schedule uses fewer buses
        if len(buses_used) < min_buses_used:
            min_buses_used = len(buses_used)
            best_schedule = current_schedule

    return best_schedule

# Generate schedule using brute force method
schedule = brute_force_schedule(manager)

# Print the generated schedule with timings and route type
print("\nGenerated Schedule (Brute Force with Route Types and Timings):")
if schedule:
    for entry in schedule:
        print(f"Bus {entry[0]} assigned to {entry[1]} ({entry[4]} Route) with Driver {entry[2]} starting at {entry[3]}")
else:
    print("No valid schedule found.")
```

✓ 0.0s

```
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 2 (Terminal) starts at 6:00
Route Route 1 (Cyclic) starts at 7:00 (PEAK HOUR)

Generated Schedule (Brute Force with Route Types and Timings):
Bus 1 assigned to Route 1 (Cyclic Route) with Driver A starting at 6:00
Bus 2 assigned to Route 2 (Terminal Route) with Driver B starting at 7:00
```

Рисунок 7- Обновленный код функцию Метод "В лоб"

```python
import random

# Функция проверки на пиковые часы
def is_in_peak_hours(start_time):
    """Проверяет, попадает ли время в пиковые часы."""
    hour = int(start_time.split(":")[0])
    for peak_start, peak_end in PEAK_HOURS:
        if peak_start <= hour < peak_end:
            return True
    return False

def genetic_algorithm_schedule(manager, generations=100, population_size=10):

    # Функция для создания случайного расписания
    def create_random_schedule():
        schedule = []
        buses_used = set()
        drivers_used = set()
        start_time = 6  # Начало расписания в 6:00
        for route in manager.routes:
            available_buses = [bus for bus in manager.buses if bus.id not in buses_used]
            available_drivers = [driver for driver in manager.drivers if driver.driver_type not in drivers_used and driver.max_hours >= route.time_required]

            if not available_buses or not available_drivers:
                print(f"Warning: No resources available for route {route.name}.")
                continue

            bus = random.choice(available_buses)
            driver = random.choice(available_drivers)

            # Определение типа маршрута
            route_type = "Cyclic" if route.is_cyclic else "Terminal"
            start_hour = f"{start_time}:00"
            peak_info = " (PEAK HOUR)" if is_in_peak_hours(start_hour) else ""

            print(f"Route {route.name} ({route_type}) starts at {start_hour}{peak_info}")

            schedule.append((bus.id, route.name, driver.driver_type, start_hour, route_type))
            buses_used.add(bus.id)
            drivers_used.add(driver.driver_type)
            start_time += 1  # Увеличиваем время старта маршрутов
        return schedule

    # Функция оценки приспособленности
    def fitness(schedule):
        return len(set(entry[0] for entry in schedule))  # Количество уникальных автобусов

    # Функция скрещивания (crossover)
    def crossover(parent1, parent2):
        split_point = len(parent1) // 2
        return parent1[:split_point] + parent2[split_point:]

    # Функция мутации (mutation)
    def mutate(schedule):
        index = random.randint(0, len(schedule) - 1)
        used_buses = {entry[0] for entry in schedule if entry != schedule[index]}
        used_drivers = {entry[2] for entry in schedule if entry != schedule[index]}

        available_buses = [bus for bus in manager.buses if bus.id not in used_buses]
        available_drivers = [driver for driver in manager.drivers if driver.driver_type not in used_drivers]

        if available_buses and available_drivers:
            bus = random.choice(available_buses)
            driver = random.choice(available_drivers)
            schedule[index] = (bus.id, schedule[index][1], driver.driver_type, schedule[index][3], schedule[index][4])

    # Генерация начальной популяции
    population = [create_random_schedule() for _ in range(population_size)]

    # Эволюция популяции
    for generation in range(generations):
        population = sorted(population, key=fitness)
        population = population[:population_size // 2]  # Отбор лучших решений

        next_generation = []
        while len(next_generation) < population_size:
            parent1, parent2 = random.sample(population, 2)
            child = crossover(parent1, parent2)
            if random.random() < 0.1:
                mutate(child)
            next_generation.append(child)

        population = next_generation

    # Возврат лучшего расписания
    best_schedule = min(population, key=fitness)
    return best_schedule
```

Рисунок 8- обновленный код Метод Генетический алгоритм

```
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)

Generated Schedule (Genetic Algorithm with Route Types and Timings):
Bus 6 assigned to Route 1 (Cyclic Route) with Driver B starting at 6:00
Bus 4 assigned to Route 2 (Terminal Route) with Driver A starting at 7:00
```

Рисунок 9-результат Метод Генетический алгоритм

5.5 результаты обоих методов :

```
Bus 1: Route None, Driver None
Bus 2: Route None, Driver None
Bus 3: Route None, Driver None
Bus 4: Route None, Driver None
Bus 5: Route None, Driver None
Bus 6: Route None, Driver None
Bus 7: Route None, Driver None
Bus 8: Route None, Driver None
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 2 (Terminal) starts at 6:00
Route Route 1 (Cyclic) starts at 7:00 (PEAK HOUR)

Generated Schedule (Brute Force):
Bus 1 assigned to Route 1 with Driver A
Bus 2 assigned to Route 2 with Driver B
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
Route Route 2 (Terminal) starts at 7:00 (PEAK HOUR)
Route Route 1 (Cyclic) starts at 6:00
...

Generated Schedule (Genetic Algorithm):
Bus 8 assigned to Route 1 with Driver B
Bus 4 assigned to Route 2 with Driver A
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Рисунок 10- результаты обоих методов

Приложения

Полный листинг кода

Определение структуры задачи

```
# Определяем класс Route (Маршрут)
class Route:
    def __init__(self, name, stops, is_cyclic, time_required):
        self.name = name  # Название маршрута
        self.stops = stops  # Остановки на маршруте
        self.is_cyclic = is_cyclic  # Является ли маршрут цикличным
        self.time_required = time_required  # Время выполнения маршрута

# Определяем класс Driver (Водитель)
class Driver:
    def __init__(self, driver_type, max_hours, breaks, shift_schedule):
        self.driver_type = driver_type  # Тип водителя: 'A' или 'B'
        self.max_hours = max_hours  # Максимальное количество рабочих часов
        self.breaks = breaks  # Перерывы водителя
        self.shift_schedule = shift_schedule  # Расписание смен водителя

# Определяем класс Bus (Автобус)
class Bus:
    def __init__(self, id, route=None, driver=None):
        self.id = id  # Идентификатор автобуса
        self.route = route  # Назначенный маршрут
        self.driver = driver  # Назначенный водитель

# Определяем ScheduleManager для управления маршрутами, автобусами и
водителями
class ScheduleManager:
    def __init__(self):
        self.routes = []  # Список маршрутов
        self.buses = []  # Список автобусов
        self.drivers = []  # Список водителей

    def add_route(self, route):
        self.routes.append(route)  # Добавление маршрута в список

    def add_bus(self, bus):
        self.buses.append(bus)  # Добавление автобуса в список

    def add_driver(self, driver):
        self.drivers.append(driver)  # Добавление водителя в список

    ef print_schedule(self):
```

```
        # Вывод расписания для всех автобусов
        for bus in self.buses:
            print(f"Автобус {bus.id}: Маршрут {bus.route.name if bus.route else 'Нет'}, Водитель
    {bus.driver.driver_type if bus.driver else 'Нет'}")
```

пример

```
if __name__ == "__main__":
    # Initialize ScheduleManager
    manager = ScheduleManager()

    # Add routes
    manager.add_route(Route("Route 1", [1, 2, 3, 2, 1], True, 60))
    manager.add_route(Route("Route 2", [1, 2, 3, 4, 5, 1], False, 70))

    # Add buses
    for i in range(1, 9):
        manager.add_bus(Bus(i))

    # Add drivers
    manager.add_driver(Driver("A", 80, [4, 1], "06:00-14:00"))  # Increased max_hours
    manager.add_driver(Driver("B", 120, [2, 15], "06:00-06:00"))

    # Print initial schedule
    manager.print_schedule()
```

Метод "Влоб"

```
        from itertools import permutations

# Определить часы пик
PEAK_HOURS = [(7, 9), (17, 19)]

def is_in_peak_hours(start_time):
    """Check if a route start time is in peak hours."""
    hour = int(start_time.split(":")[0])
    for peak_start, peak_end in PEAK_HOURS:
        if peak_start <= hour < peak_end:
            return True
    return False

def brute_force_schedule(manager):
    best_schedule = None
    min_buses_used = float('inf')

   # Сгенерировать все перестановки маршрутов
    for route_perm in permutations(manager.routes):
        buses_used = set()
        drivers_used = set()
        current_schedule = []
        start_time = 6  # Начиная с 6:00 утра

        for route in route_perm:
            route_assigned = False
            for bus in manager.buses:
                if bus.id not in buses_used:
                    for driver in manager.drivers:
                        if driver.max_hours >= route.time_required and driver.driver_type not in drivers_used:
                            start_hour = f"{start_time}:00"
                            route_type = "Cyclic" if route.is_cyclic else "Terminal"
                            peak_info = " (PEAK HOUR)" if is_in_peak_hours(start_hour) else ""

                            print(f"Route {route.name} ({route_type}) starts at {start_hour}{peak_info}")

                            # Назначить этот автобус и водителя
                            current_schedule.append((bus.id, route.name, driver.driver_type, start_hour,
route_type))
                            buses_used.add(bus.id)
                            drivers_used.add(driver.driver_type)
                            route_assigned = True
                            start_time += 1  # Увеличить время начала
                            break
```

```
                if route_assigned:
                    break

            if not route_assigned:
                print(f"Warning: No valid assignment for route {route.name}.")

        # Проверить, использует ли это расписание меньше автобусов
        if len(buses_used) < min_buses_used:
            min_buses_used = len(buses_used)
            best_schedule = current_schedule

    return best_schedule
```

метод генетический алгоритм

```
import random

# Функция проверки на пиковые часы
def is_in_peak_hours(start_time):
    """Проверяет, попадает ли время в пиковые часы."""
    hour = int(start_time.split(":")[0])
    for peak_start, peak_end in PEAK_HOURS:
        if peak_start <= hour < peak_end:
            return True
    return False

def genetic_algorithm_schedule(manager, generations=100, population_size=10):

    # Функция для создания случайного расписания
    def create_random_schedule():
        schedule = []
        buses_used = set()
        drivers_used = set()
        start_time = 6  # Начало расписания в 6:00
        for route in manager.routes:
            available_buses = [bus for bus in manager.buses if bus.id not in buses_used]
            available_drivers = [driver for driver in manager.drivers if driver.driver_type not in
drivers_used and driver.max_hours >= route.time_required]

            if not available_buses or not available_drivers:
                print(f"Warning: No resources available for route {route.name}.")
                continue

            bus = random.choice(available_buses)
            driver = random.choice(available_drivers)
```

```
            # Назначить автобус и водителя
            route_type = "Cyclic" if route.is_cyclic else "Terminal"
            start_hour = f"{start_time}:00"
            peak_info = " (PEAK HOUR)" if is_in_peak_hours(start_hour) else ""

            print(f"Route {route.name} ({route_type}) starts at {start_hour}{peak_info}")

            schedule.append((bus.id, route.name, driver.driver_type, start_hour, route_type))
            buses_used.add(bus.id)  # Mark bus as used
            drivers_used.add(driver.driver_type)  # Mark driver as used
            start_time += 1
        return schedule

    # Функция оценки приспособленности
    def fitness(schedule):
        bus_penalty = len(schedule) - len(set(entry[0] for entry in schedule))  # Penalty for reused buses
        driver_penalty = len(schedule) - len(set(entry[2] for entry in schedule))  # Penalty for reused
drivers
        return len(set(entry[0] for entry in schedule)) + bus_penalty + driver_penalty

    # Функция скрещивания (crossover)
    def crossover(parent1, parent2):
        split_point = len(parent1) // 2
        return parent1[:split_point] + parent2[split_point:]

    # Функция мутации (mutation)
    def mutate(schedule):
        index = random.randint(0, len(schedule) - 1)
        used_buses = {entry[0] for entry in schedule if entry != schedule[index]}
        used_drivers = {entry[2] for entry in schedule if entry != schedule[index]}

        available_buses = [bus for bus in manager.buses if bus.id not in used_buses]
        available_drivers = [driver for driver in manager.drivers if driver.driver_type not in used_drivers]

        if available_buses and available_drivers:
            bus = random.choice(available_buses)
            driver = random.choice(available_drivers)
            schedule[index] = (bus.id, schedule[index][1], driver.driver_type, schedule[index][3],
schedule[index][4])

    # Генерация начальной популяции
    population = [create_random_schedule() for _ in range(population_size)]

    # Эволюция популяции
    for generation in range(generations):
```

```
        population = sorted(population, key=fitness)
        population = population[:population_size // 2]  # Отбор лучших решений
        population = random.choices(population, weights=[1/fitness(s) for s in population],
k=population_size)

        next_generation = []
        while len(next_generation) < population_size:
            parent1, parent2 = random.sample(population, 2)
            child = crossover(parent1, parent2)
            if random.random() < 0.3:  # Повышенная вероятность мутации
                mutate(child)
            next_generation.append(child)

        population = next_generation

    # Возврат лучшего расписания
    best_schedule = min(population, key=fitness)
    return best_schedule
```

распечатать расписание для двух методов

```
if __name__ == "__main__":
    # Initialize ScheduleManager
    manager = ScheduleManager()

    # Add routes
    manager.add_route(Route("Route 1", [1, 2, 3, 2, 1], True, 60))
    manager.add_route(Route("Route 2", [1, 2, 3, 4, 5, 1], False, 70))

    # Add buses
    for i in range(1, 9):
        manager.add_bus(Bus(i))

    # Add drivers
    manager.add_driver(Driver("A", 80, [4, 1], "06:00-14:00"))  # Increased max_hours
    manager.add_driver(Driver("B", 120, [2, 15], "06:00-06:00"))

    # Print initial schedule
    manager.print_schedule()

    # Generate schedule using brute force method
    schedule = brute_force_schedule(manager)

    # Print the generated schedule
    print("\nGenerated Schedule (Brute Force):")
```

```
if schedule:
    for entry in schedule:
        print(f"Bus {entry[0]} assigned to {entry[1]} with Driver {entry[2]}")
else:
    print("No valid schedule found.")

# Generate schedule using genetic algorithm
ga_schedule = genetic_algorithm_schedule(manager)

# Print the generated schedule
print("\nGenerated Schedule (Genetic Algorithm):")
if ga_schedule:
    for entry in ga_schedule:
        print(f"Bus {entry[0]} assigned to {entry[1]} with Driver {entry[2]}")
else:
    print("No valid schedule found.")
```

Заключение

Оба метода работают и удовлетворяют всем ограничениям.
Метод "в лоб" обеспечивает точность, но медленный для больших задач.
Генетический алгоритм быстрее, но требует настройки и может быть менее точным.

Список литературы

- Сорокин В.В., Петров А.С. "Оптимизация транспортных систем". — Москва: Транспорт, 2018.
- Гаврилов А.В., Зайцев И.В. "Алгоритмы оптимизации: Теория и практика". — СПб: Питер, 2020.
- Документация Python (https://docs.python.org/3/).
- Официальная документация itertools (https://docs.python.org/3/library/itertools.html).
- Официальная документация random (https://docs.python.org/3/library/random.html).