УДК 004

Оптимизация распределения автобусов с помощью генетического алгоритма

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен процесс создания программы с использованием генетического алгоритма для решения задачи оптимизации управления и распределения транспортных средств. Программа написана на языке программирования Python с использованием среды программирования Google Colab. В результате исследования получено подтверждение эффективности генетического алгоритма в задачах оптимизации.
Ключевые слова: генетический алгоритм, Python, Google Colabe, алгоритм оптимизации

Optimization of bus allocation using a genetic algorithm

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article discusses the process of creating a program using a genetic algorithm to solve the problem of optimizing the management and distribution of vehicles. The program is written in the Python programming language using the Google Colab programming environment. As a result of the study, confirmation of the effectiveness of the genetic algorithm in optimization problems was obtained.
Keywords: genetic algorithm, Python, Google Colabe, optimization algorithm

1 Введение

1.1 Актуальность

В условиях современных городов, где общественный транспорт играет ключевую роль в обеспечении эффективности перемещения граждан, оптимизация управления и распределения транспортных средств становится неотъемлемой частью городского планирования. Для решения данной проблемы используются алгоритмы оптимизации, такие как градиентный спуск, RMSProp, генетический алгоритм, оптимизация Адама и многие другие. В данной статье приводится решении важной задачи оптимизации маршрутов и распределения автобусов с использованием генетического алгоритма.

1.2 Обзор исследований

Д.В. Ежов и О.В. Коваленко в своём труде рассмотрели алгоритм оптимизации «серых волков», а также реализовали улучшение данного алгоритма [1]. Д.А. Пащенко посвятил работу сравнительному исследованию двух методов оптимизации на основании результатов решения задачи минимизации массы алюминиевой фермы [2]. Н.Н. Глибовец, С.С. Гроховский и О.В. Краткова описали решения задачи оптимизации структуры интегральной схемы с помощью генетического алгоритма [3]. А.И. Замалютдинов и Л.Р. Габдрахманова провели оптимизацию параметров фотонно-кристаллического волновода с использованием генетического алгоритма [4]. А.Ю. Лабинский рассмотрел особенности использования компьютерной симуляции эволюционных процессов, часто называемых генетическими алгоритмами, для многокритериальной оптимизации [5].

1.3 Цель исследования

Целью данной статьи является описание генетического алгоритма и его реализация на примере задачи оптимизации транспортных средств.

2 Материалы и методы

Для создания программы используется язык программирования Python и библиотека matplotlib. В качестве IDE используется Google Colab.

3 Результаты и обсуждения

Генетический алгоритм представляет собой метод оптимизации и поиска, вдохновленный принципами естественного отбора и генетики. Этот алгоритм используется для решения задач оптимизации, где требуется найти наилучшее решение из множества возможных вариантов. Вот основные компоненты и шаги, которые характеризуют генетический алгоритм:

1. Инициализация популяции: на первом этапе создается начальная группа индивидов, называемая популяцией. Каждый индивид представляет собой потенциальное решение задачи.

2. Определение функции приспособленности: определяется функция, которая оценивает "приспособленность" каждого индивида в популяции. Эта функция определяет, насколько хорошо индивид соответствует целям задачи.

3. Выбор родителей: индивиды выбираются для производства потомства на основе их приспособленности. Чем более приспособлен индивид, тем больше шансов у него быть выбранным.

4. Скрещивание (кроссовер): выбранные родители комбинируют свои генетические характеристики, чтобы создать потомство. Кроссовер может осуществляться различными способами, в зависимости от задачи.

5. Мутация: случайные изменения в генетической информации происходят с определенной вероятностью. Мутации добавляют в популяцию новые варианты и могут помочь избежать застревания в локальных оптимумах.

6. Оценка потомства: приспособленность потомства оценивается с использованием той же функции приспособленности, что и у родителей.

7. Отбор (селекция): выбор индивидов, которые будут перейдены в следующее поколение. часто используются такие методы, как турнирный отбор или пропорциональный отбор, чтобы отдать предпочтение более приспособленным индивидам.

8. Повторение: процессы от 3 по 7 повторяются в течение нескольких поколений или до достижения критерия останова.

Генетические алгоритмы могут применяться к различным задачам, таким как поиск оптимальных параметров, решение задач маршрутизации, оптимизация расписания и многие другие. Они эффективны в том числе и в случаях, где простые методы оптимизации могут столкнуться с трудностями.

Сначала импортируются необходимые модули Python:

random - это встроенный модуль Python, предоставляющий функции для работы с генерацией случайных чисел. В данном случае, он может использоваться для создания случайных чисел, которые могут быть использованы в различных алгоритмах или сценариях.

matplotlib – это библиотека для создания графиков и визуализации данных на языке программирования Python.

matplotlib.pyplot - это модуль Matplotlib, который предоставляет интерфейс для создания графиков. Обычно этот модуль используется сокращенно до plt для удобства (рис. 1).

```
import random
import matplotlib.pyplot as plt
```

Рисунок 1. Импортирование модулей Python

Далее создаём начальную популяцию хромосом, представляющих различные варианты расписания движения автобусов. Функция generate_population создает начальную популяцию случайных распределений автобусов. В цикле происходит создание каждой хромосомы, представляющей распределение автобусов по номерам. Длина каждой хромосомы равна количеству автобусов (num_buses), и каждый ген хромосомы представляет номер автобуса, к которому относится определенное время ожидания (рис. 2).

```
def generate_population(population_size, num_buses):
    population = []
    for _ in range(population_size):
        chromosome = [random.randint(1, num_buses) for _ in range(num_buses)]
        population.append(chromosome)
    return population
```

Рисунок 2. Функция generate_population

Функция fitness вычисляет приспособленность каждой хромосомы, основываясь на суммарном времени ожидания для всех автобусов. Здесь

waiting_times[bus - 1] используется для получения времени ожидания конкретного автобуса в соответствии с номером из хромосомы (рис. 3).

```
def fitness(chromosome, waiting_times):
    total_waiting_time = sum(waiting_times[bus - 1] for bus in chromosome)
    return total_waiting_time
```

Рисунок 3. Функция fitness

Функции crossover и mutate реализуют операции скрещивания и мутации соответственно. В crossover случайным образом выбирается точка, где происходит обмен генами между двумя родителями. В mutate каждый ген имеет вероятность мутации (mutation_rate), и если мутация произойдет, ген изменится на случайное значение (рис. 4).

```
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(chromosome, mutation_rate):
    mutated_chromosome = []
    for gene in chromosome:
        if random.random() < mutation_rate:
            mutated_gene = random.randint(1, len(chromosome))
            mutated_chromosome.append(mutated_gene)
        else:
            mutated_chromosome.append(gene)
    return mutated_chromosome
```

Рисунок 4. Функции crossover и mutate

Функция evolve применяет элитизм, сохраняя лучшие хромосомы, и затем производит скрещивание и мутацию для создания новой популяции. Этот процесс повторяется, пока новая популяция не достигнет требуемого размера (рис. 5).

```
def evolve(population, waiting_times, mutation_rate):
    new_population = []
    sorted_population = sorted(population, key=lambda x: fitness(x, waiting_times))
    elite_size = int(0.1 * len(population))
    new_population.extend(sorted_population[:elite_size])

    while len(new_population) < len(population):
        parent1, parent2 = random.sample(sorted_population[:elite_size], 2)
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1, mutation_rate)
        child2 = mutate(child2, mutation_rate)
        new_population.extend([child1, child2])

    return new_population
```

Рисунок 5. Функция evolve

Задаём такие параметры как: количество хромосом; количество автобусов; коэффициент мутации - определяет вероятность мутации гена (номера автобуса) при каждой операции мутации; время ожидания для каждого автобуса) - представляет собой список, где каждый элемент представляет время ожидания соответствующего автобуса (рис. 6).

```
population_size = 100
num_buses = 5
mutation_rate = 0.2
waiting_times = [5, 8, 3, 10, 6]
```

Рисунок 6. Заданные параметры

В приведенном коде инициализируется популяция, затем она эволюционирует в течение нескольких поколений. После завершения эволюции, находится лучшая хромосома, которая затем визуализируется с использованием библиотеки matplotlib (рис. 7).

```
population = generate_population(population_size, num_buses)

num_generations = 50
for generation in range(num_generations):
    population = evolve(population, waiting_times, mutation_rate)

best_chromosome = min(population, key=lambda x: fitness(x, waiting_times))
plt.bar(range(1, num_buses + 1), waiting_times, label='Время ожидания без оптимизации')
plt.bar(range(1, num_buses + 1), [waiting_times[bus - 1] for bus in best_chromosome],
label='Оптимизированное время ожидания')
plt.xlabel('Номер автобуса')
plt.ylabel('Время ожидания (минуты)')
plt.legend()
plt.show()
```

Рисунок 7. Эволюция популяции и нахождение лучшей хромосомы

После выполнения программы получаем следующий результат (рис. 8).

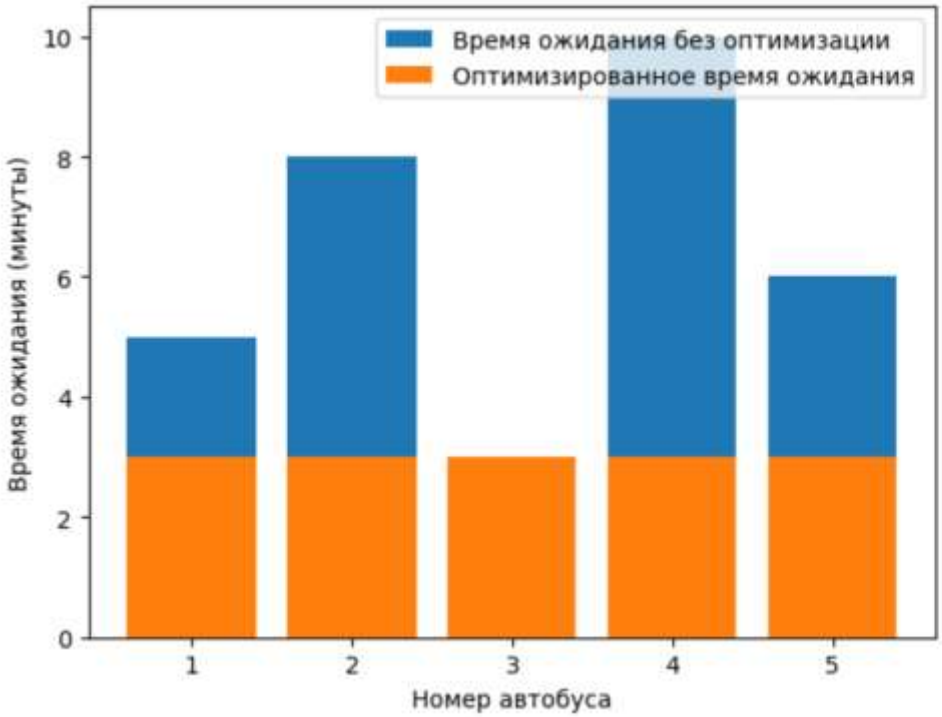

Рисунок 8. Результат оптимизации

Выводы

Программная реализация генетического алгоритма для оптимизации распределения автобусов предоставляет эффективный инструмент для управления городским транспортом. Используя данную программу, можно улучшить систему общественного транспорта, уменьшить время ожидания и повысить удовлетворенность пассажиров. Дальнейшие исследования могут сфокусироваться на тонкой настройке параметров и адаптации метода к различным городским условиям.

Библиографический список

1. Ежов Д.В., Коваленко О.В. Параллельный алгоритм оптимизации непрерывных функций большой размерности на основе гибридизации алгоритма оптимизации "серых волков" // Научный сервис в сети Интернет. 2020. № 22. С. 255-267.
2. Пащенко Д.А. Сравнение генетического алгоритма и алгоритма оптимизации роя частиц на примере одной задачи структурной оптимизации // В сборнике: Информатика, управляющие системы, математическое и компьютерное моделирование (ИУСМКМ-2020). Сборник материалов XI Международной научно-технической конференции в рамках VI Международного Научного форума Донецкой Народной Республики. Редколлегия: Ю.К. Орлов [и др.]. 2020. С. 302-306.
3. Глибовец Н.Н., Гороховский С.С., Краткова О.В. Гибридный генетический алгоритм решения задачи оптимизации структуры интегральной схемы // Разработка программного обеспечения. 2011. № 5. С. 68-76.

4. Замалютдинов А.И., Габдрахманова Л.Р. Оптимизация параметров фотонно-кристаллического волновода с использованием генетического алгоритма // В сборнике: Перспективные информационные технологии (ПИТ 2014). труды Международной научно-технической конференции. 2014. С. 40-42.
5. Лабинский А.Ю. Использование генетического алгоритма для многокритериальной оптимизации // Природные и техногенные риски (физико-математические и прикладные аспекты). 2018. № 4 (28). С. 5-9.