ҚАЗАҚСТАН РЕСПУБЛИКАСЫ ҒЫЛЫМ ЖӘНЕ ЖОҒАРЫ БІЛІМ МИНИСТРЛІГІ

«Л.Н. ГУМИЛЕВ АТЫНДАҒЫ ЕУРАЗИЯ ҰЛТТЫҚ УНИВЕРСИТЕТІ» КЕАҚ

Студенттер мен жас ғалымдардың
«ǴYLYM JÁNE BILIM - 2024»
XIX Халықаралық ғылыми конференциясының
БАЯНДАМАЛАР ЖИНАҒЫ

СБОРНИК МАТЕРИАЛОВ
XIX Международной научной конференции
студентов и молодых ученых
«ǴYLYM JÁNE BILIM - 2024»

PROCEEDINGS
of the XIX International Scientific Conference
for students and young scholars
«ǴYLYM JÁNE BILIM - 2024»

2024
Астана

УДК 001
ББК 72
G99

«ǴYLYM JÁNE BILIM – 2024» студенттер мен жас ғалымдардың XIX Халықаралық ғылыми конференциясы = XIX Международная научная конференция студентов и молодых ученых «ǴYLYM JÁNE BILIM – 2024» = The XIX International Scientific Conference for students and young scholars «ǴYLYM JÁNE BILIM – 2024». – Астана: – 7478 б. - қазақша, орысша, ағылшынша.

ISBN 978-601-7697-07-5

Жинаққа студенттердің, магистранттардың, докоранттардың және жас ғалымдардың жаратылыстану-техникалық және гуманитарлық ғылымдардың өзекті мәселелері бойынша баяндамалары енгізілген.

The proceedings are the papers of students, undergraduates, doctoral students and young researchers on topical issues of natural and technical sciences and humanities.

В сборник вошли доклады студентов, магистрантов, докторантов имолодых ученых по актуальным вопросам естественно-технических и гуманитарных наук.

УДК 001
ББК 72
G99

ISBN 978-601-7697-07-5

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ РЕШЕНИЯ ЗАДАЧИ ДОСТАВКИ ГРУЗОВ

Бисенкул Алмас Сабыржанулы
almasbisenkul7@gmail.com
магистрант, кафедра «Информационные системы», Евразийский Национальный Университет им. Л.Н. Гумилева, Казахстан, г. Астана
Научный руководитель – Ла Лира Львовна

Проблема доставки грузов является актуальной задачей и для ее решения существует различные подходы, такие как например, метод линейного программирования. Одним из способов решения данной задачи является применение генетических алгоритмов, позволяющих найти оптимальное решение за достаточно приемлемое время. В данной работе рассматривается одна модель задачи доставки грузов и разработан генетический алгоритм (ГА) ее решения. Предлагается реализация разработанного алгоритма на языке Python. Проблема доставки грузов аналогична задаче коммивояжера и описывается следующим образом:

"Учитывая список городов и расстояния между городов, каков самый короткий возможный маршрут, который посещает каждый город и возвращается в исходный город?"

Исходя из этого, стоит помнить о двух важных правилах:

1.Каждый город должен быть посещен ровно один раз

2.Мы должны вернуться в начальный город, поэтому общее расстояние должно быть рассчитано соответственно

Начнем с нескольких определений:

Ген: город (представленный координатами (x, y))

Индивид (также "хромосома"): один маршрут, удовлетворяющий вышеуказанным условиям

Популяция: набор возможных маршрутов (т.е. набор индивидов)

Родители: два маршрута, которые объединяются для создания нового маршрута

Парный пул: набор родителей, используемых для создания следующей популяции (тем самым создающих следующее поколение маршрутов)

Приспособленность: функция, которая говорит нам, насколько хорош каждый маршрут (в нашем случае, насколько короткое расстояние)

Мутация: способ внесения изменений в нашу популяцию путем случайного обмена двух городов в маршруте

Элитизм: способ переноса лучших индивидов в следующее поколение

ГА будет работать следующим образом:

1. Создать популяцию
2. Определить приспособленность
3. Выбрать парный пул
4. Скрещиваться
5. Мутировать
6. Повторять

Мы будем использовать несколько стандартных пакетов, чтобы упростить задачу:

```
import numpy as np, random, operator, pandas as pd, matplotlib.pyplot
as plt
```

Рисунок 1. Подключение стандартных пакетов

Создаем два класса: City и Fitness.

Сначала мы создаем класс City, который позволит нам создавать и управлять нашими городами. Это просто наши координаты (x, y). Внутри класса City мы добавляем вычисление расстояния (используя теорему Пифагора) и способ вывода городов как координат с помощью метода repr.

```
class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, city):
        xDis = abs(self.x - city.x)
        yDis = abs(self.y - city.y)
        distance = np.sqrt((xDis ** 2) + (yDis ** 2))
        return distance

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"
```

Рисунок 2. Создание класса City

Мы также создадим класс Fitness. В нашем случае мы будем рассматривать фитнес как обратное расстояние маршрута. Мы хотим минимизировать расстояние маршрута, поэтому более высокий балл фитнеса - это лучше. Исходя из Правила №2, мы должны начинать и заканчивать в одном и том же месте.

```
class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness= 0.0

    def routeDistance(self):
        if self.distance ==0:
            pathDistance = 0
            for i in range(0, len(self.route)):
                fromCity = self.route[i]
                toCity = None
                if i + 1 < len(self.route):
                    toCity = self.route[i + 1]
                else:
                    toCity = self.route[0]
                pathDistance += fromCity.distance(toCity)
            self.distance = pathDistance
        return self.distance

    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
```

Рисунок 3. Создание класса Fitness

Создание популяции

Теперь мы можем создать нашу начальную популяцию . Для этого нам нужен способ создать функцию, которая генерирует маршруты, удовлетворяющие нашим условиям. Чтобы создать отдельного индивида, мы случайным образом выбираем порядок посещения каждого города:

```
def createRoute(cityList):
    route = random.sample(cityList, len(cityList))
    return route
def initialPopulation(popSize, cityList):
    population = []

    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population
```

Рисунок 4. Создание популяции

Выбор пула родителей

Существует несколько вариантов того, как выбирать родителей, которые будут использоваться для создания следующего поколения. Самые распространенные подходы - это либо выбор с пропорциональностью фитнеса (так называемый "выбор на основе рулетки") или турнирный выбор:

Турнирный выбор: Из популяции случайным образом выбирается определенное количество индивидов, и тот, у кого наивысший фитнес в группе, выбирается первым родителем. Это повторяется для выбора второго родителя.

Еще одна особенность, которую стоит учитывать, это использование элитарности. С элитарностью лучшие по результатам индивиды из популяции автоматически переходят в следующее поколение, обеспечивая сохранение наиболее успешных индивидов.

Для большей ясности мы создадим пул родителей в два этапа. Сначала мы будем использовать результат rankRoutes, чтобы определить, какие маршруты выбрать в нашей функции выбора. Мы настраиваем рулетку, вычисляя относительный вес фитнеса для каждого индивида. Мы сравниваем случайное число с этими весами, чтобы выбрать наш пул родителей. Мы также хотим сохранить наши лучшие маршруты, поэтому мы вводим элитарность. В конечном итоге функция выбора возвращает список идентификаторов маршрутов, которые мы можем использовать для создания пула родителей в функции matingPool.

```
def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults
```

Рисунок 5. Создание турнирного выбора

Вводим изменений в нашу популяцию путем случайного обмена двух городов в маршруте

```
def mutate(individual, mutationRate):
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1
    return individual
```

Рисунок 6. Создание мутации

Запуск генетического алгоритма

```
geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=500)
```

Рисунок 7. Запуск генетического алгоритма

Результат

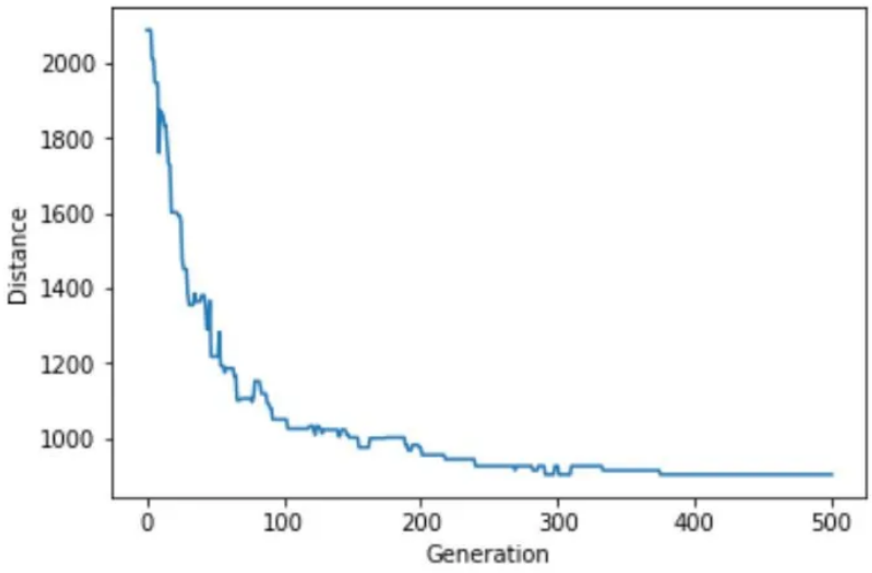

Рисунок 8. Результат

Исходя из результатов можно уверенностью сказать что при увеличении количества потомков в популяции наблюдается сокращение дистанции между городами на найденном маршруте. Это говорит о том, что увеличение размера популяции способствует более эффективному поиску оптимального маршрута, так как большее количество потомков позволяет алгоритму проводить более глубокий и широкий поиск возможных решений.

Список использованных источников

1. Holland, John H. Adaptation in Natural and Artificial Systems. // MIT Press, 1992, P. 198-205

2. Mitchell, Melanie. An Introduction to Genetic Algorithms. // MIT Press, 1996, P. 837-840

3.Davis, Lawrence. Handbook of Genetic Algorithms. // Van Nostrand Reinhold, 1991, P. 65-78

4.Eiben, Agoston E., and Smith, James E. Introduction to Evolutionary Computing. // Springer, 2015, P. 238-240

УДК 004.94

ИССЛЕДОВАНИЕ НЕЙРОДЕГЕНЕРАТИВНЫХ ЗАБОЛЕВАНИЙ. С ИСПОЛЬЗОВАНИЕМ КОМПЬЮТЕРНОГО ЗРЕНИЯ

Дакенов Алишер Мырзахметұлы
trapbeztrapbez@gmail.com
Студент- Факультета информационных технологий, кафедра(Технология Исскуственного Интеллекта),Ену им Л.Н. Гумилева г. Астана, Казахстан
Научный руководитель - Садвакасов Р.М.

Аннотация: В современном медицинском обществе заболевания головного мозга, такие болезни как инсульт, Паркинсона и Альцгеймера, представляют собой глобальную проблему, затрагивающую не только отдельных пациентов, но и всю их социальную среду. Эти состояния, постепенно ограничивая когнитивные и моторные функции, серьезно ухудшают качество жизни тех, кто сталкивается с ним. Они оказывают значительное воздействие на способность общения, самообслуживания и участия в повседневных активностях, создавая для пациентов и их близких ряд трудностей в поддержании привычного образа жизни и социальной активности.[1,2]

Ключивые слова: заболевания головного мозга, компьютерное зрение, глубокое обучение, сверточные автоэнкодеры, автоматическая диагностика, точность.

Введение: С целью улучшения диагностики, мониторинга и лечения таких заболеваний в современных медицинских исследованиях наблюдается увеличенный интерес к разработке новых подходов. Основная цель этих исследований заключается в поиске более эффективных методов диагностики и мониторинга состояния пациентов с заболеваниями головного мозга. В этом контексте компьютерное зрение, подкрепленное современными алгоритмами обработки данных и машинным обучением, становится важным инструментом исследования. Оно открывает новые перспективы в области диагностики и мониторинга состояния пациентов, а также способно повысить эффективность терапевтических вмешательств. Таким образом, использование передовых технологий в медицинских исследованиях становится необходимым шагом в направлении улучшения прогноза и качества жизни пациентов с заболеваниями головного мозга. Основываясь на зарубежные источники в области информационных технологий.

В работе автора Yagis Ekin [3] представляет новый метод автоматической диагностики болезни Альцгеймера (AD) на основе глубокого обучения с использованием сверточных автоэнкодеров (CAE). Ииспользуемые методы и подходы: Для автоматической диагностики болезни Альцгеймера применялись сверточные автоэнкодеры (CAE) с 26 слоями для извлечения низкоразмерных представлений данных МРТ мозга. Они объединяли надзорное и ненадзорное обучение, что позволило извлечь признаки из структурных МРТ-сканов мозга. Оценка модели включала расчет среднеквадратичной ошибки (MSE), отношения сигнал-шум (PSNR), точности и F1-меры. преимущество: Разработанный метод демонстрирует высокую точность классификации болезни Альцгеймера, превосходя традиционные классификаторы, даже при использовании только одного среза МРТ. Кроме того, он способен обнаруживать паттерны церебральной атрофии, что помогает выявить ранние изменения в мозге,