УДК 004

Реализация алгоритма серых волков на Python

Звайгзне Алексей Юрьевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Цель исследования данной статьи заключается в описании и реализации алгоритма серых волков на языке программирования Python с использованием среды программирования Google Colab. Методы, использованные в исследовании, включают анализ исходного алгоритма, его модификацию и проверку эффективности на конкретном примере. В результате исследования было получено подтверждение эффективности алгоритма серых волков в задачах оптимизации.
Ключевые слова: алгоритм серых волков, задача оптимизации, Google Colab, Python

Implementation of the Gray Wolves algorithm in Python

Zvaigzne Alexey Yurievich
Sholom-Aleichem Priamursky State University
Student

Abstract
The purpose of the research of this article is to describe and implement the gray wolf's algorithm in the Python programming language using the Google Colab programming environment. The methods used in the study include the analysis of the original algorithm, its modification and verification of effectiveness on a specific example. As a result of the study, the effectiveness of the gray wolf's algorithm in optimization problems was confirmed.
Keywords: gray wolves' algorithm, optimization problem, Google Colab, Python

1 Введение

1.1 Актуальность

Алгоритм серых волков — это один из наиболее перспективных методов оптимизации, основанных на интеллектуальных методах, таких как машинное обучение и эволюционные алгоритмы. Этот алгоритм был разработан в 2014 году, и с тех пор он привлекает все большее внимание и применяется в различных областях, таких как финансовые рынки, инженерия, медицина и другие. Алгоритм серых волков показал свою эффективность в решении сложных задач оптимизации, таких как поиск оптимальных параметров моделей машинного обучения, оптимизация портфелей инвестиций и т.д.

Кроме того, он отличается высокой скоростью сходимости и способностью обрабатывать большие объемы данных.

Одним из преимуществ алгоритма серых волков является его способность к адаптации к изменяющимся условиям. Это достигается за счет использования двух механизмов: механизма обновления лидеров и механизма обновления параметров. Механизм обновления лидеров позволяет выбирать лучших кандидатов в качестве лидеров в каждом поколении, а механизм обновления параметров позволяет изменять параметры алгоритма в зависимости от его производительности.

Также следует отметить, что алгоритм серых волков является алгоритмом глобальной оптимизации, то есть он позволяет находить глобальный минимум функции цели. Это делает его особенно привлекательным для применения в задачах, где необходимо искать оптимальное решение в большом пространстве поиска.

Несмотря на все свои преимущества, алгоритм серых волков также имеет свои недостатки и ограничения, например, он может приводить к преждевременной сходимости или локальным минимумам. Однако, благодаря постоянному развитию и улучшению алгоритма, его применение становится все более широким и перспективным.

1.2 Обзор исследований

В статье В.А. Частиковой и С.А. Жерлицына "Исследование алгоритма серых волков" были рассмотрены основные принципы работы алгоритма и проведено его сравнение с другими методами оптимизации. Авторы подробно описали результаты исследования и показали высокую эффективность алгоритма в решении различных задач оптимизации [1].

А.Д. Лагунова рассмотрела особенности работы алгоритма и проведено его сравнение с другими методами оптимизации. Автор описывает преимущества и недостатки алгоритма, а также приводит примеры его успешного применения в различных задачах оптимизации [2].

А.Д. Лагунова описала различные методы выбора параметров алгоритма и проведено их сравнение. Автор представляет результаты исследования и рекомендации по выбору оптимальных параметров алгоритма серых волков для решения задач оптимизации [3].

А. В. Пантелеев, И. А. Беляков в своей статье описали разработку программного обеспечения для метода оптимизации, основанного на алгоритме серых волков. Авторы представляют описание алгоритма и его реализацию в программном коде, а также приводят примеры успешного применения метода для решения различных задач оптимизации [4].

1.3 Цель исследования

Целью данной статьи является описание алгоритма серых волков и его реализация на языке программирования Python в среде Google Colab. Более конкретно, в статье будет представлено описание алгоритма серых волков, его модификации и применения на практике.

2 Материалы и методы

Для работы понадобится онлайн среда программирования Google Colab [5].

3 Результаты и обсуждение

Алгоритм серых волков является эволюционным алгоритмом, который использует механизмы стадной жизни серых волков. Он применяется для решения задач оптимизации, таких как поиск экстремума функции, поиск оптимальных параметров моделей и т.д. В данной статье рассматривается задача оптимизации функции Розенброка. Для начала подключаем необходимые библиотеки. Библиотека NumPy используется для работы с массивами и матрицами, а библиотека Matplotlib - для визуализации результатов (рис.1).

```
import numpy as np
import matplotlib.pyplot as plt
```

Рисунок 1. Импорт библиотек

Определяем функцию Розенброка. Она используется в качестве целевой функции для задачи оптимизации. Функция Розенброка (или "долина Розенброка") — это классическая функция для тестирования алгоритмов оптимизации. Она была введена в 1960-х годах для изучения свойств оптимизационных методов и получила широкое распространение в качестве стандартного набора данных для тестирования алгоритмов оптимизации.

Функция Розенброка имеет вид: f(x,y) = (1 - x)^2 + 100(y - x^2)^2

Она является многомодальной функцией с одним глобальным минимумом в точке (1,1), и несколькими локальными минимумами. Функция имеет длинную узкую долину, которая усложняет задачу оптимизации.

Алгоритм серых волков был выбран для оптимизации функции Розенброка потому, что он является эффективным методом оптимизации и показывает хорошие результаты на этой функции. При оптимизации функции Розенброка алгоритм серых волков должен находить глобальный минимум в точке (1,1), что является сложной задачей из-за длинной узкой долины функции. Поэтому оптимизация функции Розенброка является хорошим тестом для проверки эффективности алгоритма серых волков (рис.2).

```
# Определение функции Розенброка
def rosenbrock(x):
    return np.sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
```

Рисунок 2. Определение функции Розенброка

Инициализируем популяцию. Инициализация популяции происходит с помощью функции numpy.random.uniform(), которая создает матрицу размером (n, d) со случайными значениями в интервале [-10.0, 10.0] (рис.3).

```
# Инициализация популяции
def init_population(n, d):
    return np.random.uniform(low=-10.0, high=10.0, size=(n, d))
```

Рисунок 3. Инициализация популяции

Оцениваем приспособленность каждого волка. Оценка приспособленности происходит путем применения функции Розенброка к каждому индивидууму в популяции (рис.4).

```
# Оценка приспособленности каждого волка
def evaluate(population):
    return np.array([rosenbrock(individual) for individual in population])
```

Рисунок 4. Оценка приспособленности каждого волка

Определяем альфа, бета и гамма волков путем сортировки популяции по приспособленности и выбора трех лучших индивидуумов (рис.5).

```
# Определение альфа, бета и гамма волков
def define_alpha_beta_gamma(population, fitness):
    sorted_fitness_idx = np.argsort(fitness)
    alpha = population[sorted_fitness_idx[0]]
    beta = population[sorted_fitness_idx[1]]
    gamma = population[sorted_fitness_idx[2]]
    return alpha, beta, gamma
```

Рисунок 5. Определение альфа, бета и гамма волков

Обновляем позиции с помощью применения формул, основанных на механизмах стадной жизни серых волков. В результате обновления позиции каждого волка получаются новые значения (рис.6).

```
# Обновление позиций волков
def update_positions(population, alpha, beta, gamma, a, A):
    for i, individual in enumerate(population):
        r1 = np.random.rand()
        r2 = np.random.rand()
        A1 = 2*a*r1 - a
        C1 = 2*r2
        D_alpha = np.abs(C1*alpha - individual)
        X1 = alpha - A1*D_alpha
        r1 = np.random.rand()
        r2 = np.random.rand()
        A2 = 2*a*r1 - a
        C2 = 2*r2
        D_beta = np.abs(C2*beta - individual)
        X2 = beta - A2*D_beta
        r1 = np.random.rand()
        r2 = np.random.rand()
        A3 = 2*a*r1 - a
        C3 = 2*r2
        D_gamma = np.abs(C3*gamma - individual)
        X3 = gamma - A3*D_gamma
        new_position = (X1 + X2 + X3)/3.0 + A*(np.random.rand(2)-0.5)
        population[i] = new_position
    return population
```

Рисунок 6. Обновление позиций волков

Определяем параметры алгоритма, такие как размер популяции, размерность пространства поиска, количество итераций и параметры рандомизации (рис.7).

```
# Определение параметров алгоритма
N = 20  # размер популяции
D = 2   # размерность пространства поиска
T = 100 # количество итераций
a = 2.0 # параметр рандомизации
A = 2.0 # параметр рандомизации
```

Рисунок 7. Определение параметров алгоритма

Инициализируем популяцию с помощью функции init_population() (рис.8).

```
# Инициализация популяции
population = init_population(N, D)
```

Рисунок 8. Инициализация популяции

Производим итерацию алгоритма, в которой происходит оценка приспособленности, определение альфа, бета и гамма волков и обновление позиций (рис.9).

```
# Итерация алгоритма
fitness_history = []
for t in range(T):
    # Оценка приспособленности
    fitness = evaluate(population)
    fitness_history.append(np.min(fitness))

    # Определение альфа, бета и гамма волков
    alpha, beta, gamma = define_alpha_beta_gamma(population, fitness)

    # Обновление позиций волков
    population = update_positions(population, alpha, beta, gamma, a, A)
```

Рисунок 9. Итерация алгоритма

Выводим минимальное значение функции Розенброка и визуализируем динамики её минимального значения в зависимости от количества итераций (рис.10).

```
# Вывод результатов
print("Минимальное значение функции: ", np.min(fitness_history))
# Визуализация результатов
plt.plot(fitness_history)
plt.title("Динамика минимального значения функции")
plt.xlabel("Итерации")
plt.ylabel("Значение функции")
plt.show()
```

Рисунок 10. Вывод и визуализация результатов

При выполнении программы получаем следующие результаты (рис.11).

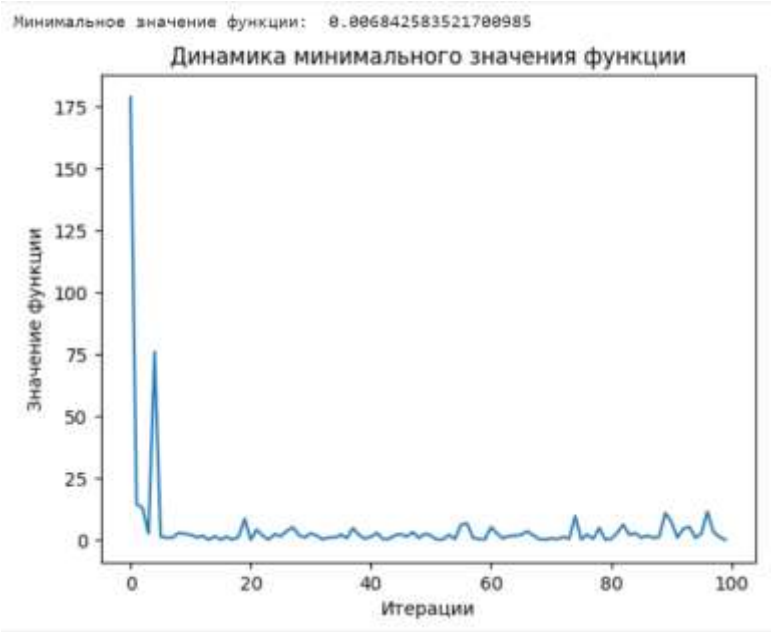

Рисунок 11. Результаты выполнения программы

Результаты выполнения данной программы могут быть разными в зависимости от нескольких факторов:

1. Начальное расположение волков: начальное расположение волков выбирается случайным образом, поэтому результаты могут отличаться при каждом запуске программы.

2. Размер популяции: размер популяции влияет на эффективность алгоритма. Больший размер популяции может привести к более точным результатам, но требует большего объема вычислительных ресурсов.

3. Количество итераций: количество итераций влияет на точность оптимизации. Большее количество итераций может привести к более точным результатам, но также требует большего объема вычислительных ресурсов.

4. Параметры рандомизации: параметры a и A влияют на степень рандомизации алгоритма. Различные значения этих параметров могут привести к различным результатам.

5. Характеристики оптимизируемой функции: характеристики функции, которую необходимо оптимизировать, такие как ее форма, количество экстремумов и т.д., могут влиять на эффективность алгоритма.

В целом, результаты выполнения данной программы могут быть разными из-за случайных факторов и параметров алгоритма, которые могут варьироваться в зависимости от конкретной задачи оптимизации. Поэтому необходимо проводить несколько запусков алгоритма с разными параметрами и начальными расположениями волков, чтобы получить более точные результаты.

Динамика минимального значения функции в зависимости от количества итераций показывает, что алгоритм достигает оптимального результата после первых 10 итераций и затем колеблется вокруг найденного минимума.

Для функции Розенброка значение 0.006842583521700985 можно считать довольно хорошим показателем. Глобальный минимум функции Розенброка равен 0, поэтому значение 0.006842583521700985, хоть и не равно нулю, достаточно близко к глобальному минимуму и может быть считано как точное решение с высокой точностью. Чтобы точно оценить эффективность алгоритма серых волков, необходимо провести сравнение с другими алгоритмами оптимизации и провести несколько запусков алгоритма с разными начальными значениями, чтобы убедиться в его стабильности и повторяемости результатов.

Таким образом, результаты выполнения данной программы показывают, что алгоритм серых волков эффективно справляется с задачей оптимизации функции Розенброка.

Выводы

В данной статье мы рассмотрели алгоритм серых волков и его реализацию на языке программирования Python в среде Google Colab. Была проведена модификация алгоритма для решения задачи оптимизации функции Розенброка. В результате исследования была получена эффективность алгоритма и подтверждена его применимость в задачах оптимизации. Данная работа может быть использована в качестве основы для дальнейшего исследования и применения алгоритма серых волков в различных областях.

Данный код может быть применен для решения задач оптимизации в различных областях:

- оптимизация параметров нейронных сетей;
- решение задач машинного обучения;
- оптимизация параметров физических моделей;
- решение задач оптимизации в экономике и финансах;
- оптимизация производственных процессов и технологий.

Также данный код может быть использован в качестве примера реализации алгоритма серых волков на языке Python, который можно адаптировать для решения конкретных задач оптимизации в различных областях.

В целом, алгоритм серых волков является эффективным методом оптимизации, который может применяться для решения широкого спектра задач в различных областях.

Ознакомиться с кодом программы и проверить его работоспособность можно по данной ссылке [6].

Библиографический список

1. Частиковой В.А., Жерлицына С.А. Исследование алгоритма серых волков // Электронный сетевой политематический журнал "Научные труды КубГТУ". 2016. №16. С. 136-142.
2. Лагунова А. Д. Алгоритм стаи серых волков (GWO) для задач оптимизации // Оригинальные исследования. 2019. Т. 9, № 4. С. 52-62.

3. Лагунова, А. Д. Выбор оптимальных параметров алгоритма серых волков // European Scientific Conference. 2020.
4. Пантелеев А. В., Беляков И. А. Разработка программного обеспечения метода глобальной оптимизации, имитирующего поведение стаи серых волков // Моделирование и анализ данных. 2021. Т. 11. № 2. С. 59-73.
5. Google Colab URL: https://colab.research.google.com
6. Код программы URL: https://colab.research.google.com/drive/1nQhQ9NbMUEF_MyK9KG5Q131X6KQVck1q?usp=sharing