УДК 004

Применение муравьиного алгоритма

Романов Даниил Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Цель данной статьи заключается в применении муравьиного алгоритма для решения задачи коммивояжера, которая является одной из классических задач комбинаторной оптимизации. В исследовании рассмотрен муравьиный алгоритм, написанный на языке программирования Python в среде программирования Google Colab. В результате исследования показано, что муравьиный алгоритм и его модификации могут быть эффективными инструментами для решения задач комбинаторной оптимизации.
Ключевые слова: муравьиный алгоритм, задача коммивояжера, комбинаторная оптимизация, Python, Google Colab

Application of the ant algorithm

Romanov Daniil Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The purpose of this article is to apply the ant algorithm to solve the traveling salesman problem, which is one of the classical combinatorial optimization problems. The study examined an ant algorithm written in the Python programming language in the Google Colab programming environment. As a result of the study, it was shown that the ant algorithm and its modifications can be effective tools for solving combinatorial optimization problems.
Keywords: ant algorithm, traveling salesman problem, combinatorial optimization, Python, Google Colab

1 Введение

1.1 Актуальность

В современном мире все большее значение приобретают задачи оптимизации, которые решаются с помощью различных алгоритмов. Одним из наиболее эффективных и популярных является муравьиный алгоритм, который основан на поведении муравьев при поиске кратчайшего пути до источника питания. Муравьиный алгоритм применяется во многих областях, таких как транспорт, логистика, телекоммуникации, финансы и т.д. Он позволяет решать задачи маршрутизации, планирования производства, оптимизации портфеля инвестиций и многие другие.

1.2 Обзор исследований

С. Д. Штовба описала муравьиный алгоритм и его применение для решения различных задач оптимизации. В её статье рассматриваются основные принципы работы алгоритма, а также приводятся примеры его использования в задачах коммивояжера и планирования производства [1].

В. М. Курейчика и А. А. Кажарова посвятили свою статью различным модификациям муравьиного алгоритма, таким как алгоритм муравьиной колонии, муравьиная система с обратной связью и другие. В статье описываются основные принципы работы каждой модификации и приводятся результаты исследований их эффективности в решении различных задач оптимизации [2].

А. А. Кажарова и В. М. Курейчика в своём исследовании описывают применение муравьиного алгоритма для решения задач транспортной логистики. В статье рассматриваются особенности задач транспортной логистики и приводятся примеры применения муравьиного алгоритма для их решения [3].

1.3 Цель исследования

Цель данного исследования - изучение принципов работы муравьиного алгоритма и его применение для решения задачи коммивояжера. Для достижения этой цели написана программа на языке программирования Python, реализующая муравьиный алгоритм и позволяющая решать задачу коммивояжера.

2 Материалы и методы

Для работы понадобится онлайн среда программирования Google Colab [4].

3 Результаты и обсуждение

Задача коммивояжера - это задача поиска кратчайшего пути, проходящего через все заданные города и возвращающегося в начальный город. Для решения этой задачи можно использовать муравьиный алгоритм. Для применения муравьиного алгоритма для решения задачи коммивояжера необходимо представить города в виде графа, где вершины - это города, а ребра - расстояния между городами. Затем необходимо распределить феромоны на ребрах графа и запустить муравьиный алгоритм. Приступаем к созданию класса AntColony, который содержит все необходимые методы для работы с муравьиным алгоритмом. Для начала импортируем библиотеку numpy, которая используется для работы с массивами и матрицами (рис.1).

```
import numpy as np
```

Рисунок 1 - Импорт библиотеки numpy

Определяем класс AntColony, который содержит все необходимые методы для работы с муравьиным алгоритмом. В методе __init__

инициализируются все необходимые параметры для работы алгоритма, такие как матрица расстояний distances, количество муравьев n_ants, количество итераций алгоритма n_iterations, скорость испарения феромонов decay, а также параметры alpha и beta, которые используются для настройки эвристической функции. Кроме того, в этом методе создается матрица феромонов pheromone, которая инициализируется равномерно распределенными значениями (рис.2).

```
class AntColony(object):
    def __init__(self, distances, n_ants, n_iterations, decay, alpha=1, beta=1):
        self.distances = distances
        self.n_ants = n_ants
        self.n_iterations = n_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta
        self.pheromone = np.ones(distances.shape) / len(distances)
```

Рисунок 2 - Создание класса AntColony

Создаём метод run, который запускает муравьиный алгоритм и возвращает лучшее найденное решение. На каждой итерации алгоритма генерируются пути с помощью метода generate_paths, затем обновляются феромоны на основе найденных путей с помощью метода update_pheromone. Далее вычисляются длины всех путей с помощью метода path_distance, выбирается лучший путь и обновляется матрица феромонов. После завершения всех итераций алгоритма возвращается лучший путь и его длина (рис.3).

```
    def run(self):
        best_distance = np.inf
        best_path = []

        for i in range(self.n_iterations):
            paths = self.generate_paths()
            self.update_pheromone(paths)
            distances = [self.path_distance(path) for path in paths]
            best_index = np.argmin(distances)
            if distances[best_index] < best_distance:
                best_distance = distances[best_index]
                best_path = paths[best_index]
            self.pheromone *= self.decay

        return best_path, best_distance
```

Рисунок 3 - Создание метода run

Метод generate_paths генерирует заданное количество путей с помощью метода generate_path (рис.4).

```
    def generate_paths(self):
        paths = []
        for i in range(self.n_ants):
            path = self.generate_path()
            paths.append(path)
        return paths
```

Рисунок 4 - Создание метода generate_paths

Метод generate_path генерирует один путь, используя жадный алгоритм и эвристическую функцию на основе феромонов и расстояний. Сначала выбирается случайная начальная вершина, затем на каждом шаге выбирается следующая вершина на основе значения феромонов и расстояний. Вершины, которые уже были посещены, не учитываются (рис.5).

```
def generate_path(self):
  path = []
  visited = set()
  current = np.random.randint(len(self.distances))
  visited.add(current)
  path.append(current)
  while len(visited) < len(self.distances):
      pheromones = self.pheromone[current]
      distances = self.distances[current]
      unvisited = set(range(len(distances))) - visited
      unvisited_arr = np.array(list(unvisited))
      if len(unvisited) == 0:
          break
      numerator = np.power(pheromones[unvisited_arr], self.alpha) * np.power(1 / distances[unvisited_arr], self.beta)
      probabilities = numerator / np.sum(numerator)
      next_index = np.random.choice(unvisited_arr, p=probabilities)
      visited.add(next_index)
      path.append(next_index)
      current = next_index
  return path
```

Рисунок 5 - Создание метода generate_path

Метод update_pheromone обновляет значения феромонов на основе всех найденных путей. Для каждой пары вершин вычисляется изменение феромонов, которое зависит от длины пути и того, сколько муравьев прошли по этому пути. Затем значения феромонов обновляются с учетом скорости испарения (рис.6).

```
def update_pheromone(self, paths):
    for i in range(len(self.distances)):
        for j in range(len(self.distances)):
            if i != j:
                delta_pheromone = 0
                for path in paths:
                    if (i, j) in zip(path, path[1:]):
                        delta_pheromone += 1 / self.path_distance(path)
                self.pheromone[i, j] = self.pheromone[i, j] * self.decay + delta_pheromone
```

Рисунок 6 - Создание метода update_pheromone

Метод path_distance вычисляет длину заданного пути, используя матрицу расстояний (рис.7).

```
def path_distance(self, path):
    distance = 0
    for i in range(len(path) - 1):
        distance += self.distances[path[i], path[i+1]]
    distance += self.distances[path[-1], path[0]]
    return distance
```

Рисунок 7 - Создание метода path_distance

Чтобы проверить работоспособность кода создаём матрицу расстояний distances, которая задает расстояния между каждой парой вершин в графе.

Затем создаём объект класса AntColony с заданными параметрами: матрицей расстояний distances, количеством муравьев n_ants равным 10, количеством итераций n_iterations равным 100 и скоростью испарения феромонов decay равной 0.5. Затем запускаем метод run для объекта colony, который находит лучший путь и его длину. Полученные результаты выводятся на экран с помощью функции print. Выводится лучший найденный путь best_path и его длина best_distance (рис.8).

```
distances = np.array([[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]])
colony = AntColony(distances, n_ants=10, n_iterations=100, decay=0.5)
best_path, best_distance = colony.run()
print("Best path:", best_path)
print("Best distance:", best_distance)
```

Рисунок 8 - Проверка работоспособности

После запуска программы получаем следующие результаты (рис.9).

```
Best path: [3, 1, 0, 2]
Best distance: 80
```

Рисунок 9 - Результат программы

Выводы

В результате исследования было установлено, что муравьиный алгоритм является эффективным инструментом для решения задач оптимизации, в том числе задачи коммивояжера. Программа на языке Python, реализующая муравьиный алгоритм, позволяет решать задачи оптимизации с высокой точностью и скоростью. Поэтому муравьиный алгоритм может быть рекомендован для применения в различных областях, где требуется решение задач оптимизации. ознакомиться с кодом и проверить его работоспособность можно по данной ссылке [5].

Библиографический список

1. Штовба С.Д. Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. 2003. С. 70-75.
2. Курейчик В.М., Кажаров А.А. О некоторых модификациях муравьиного алгоритма // Известия Южного федерального университета. Технические науки. 2008. Т. 81. № 4. С. 7-12.
3. Кажаров А.А., Курейчик В.М. Муравьиные алгоритмы для решения транспортных задач // Известия Российской академии наук. Теория и системы управления. 2010. № 1. С. 32-45.
4. Google Colab URL: https://colab.research.google.com
5. Код программы URL: https://colab.research.google.com/drive/1-aoRo-IZSqtIxbQfig93WAKKdGauN15Q?usp=sharing