ВВЕДЕНИЕ

Задача коммивояжера является одной из самых известных задач комбинаторной оптимизации, заключается в поиске самого выгодного цикла графа (замкнутого маршрута), проходящего через все его вершины. Впервые задача была сформулирована как математическая задача в 1930 году Карлом Менгером (1903–1985), хотя сама проблема с практической точки зрения была описана ещё в 1832 году.

Задача относится к трансвычислительным задачам, количество возможных циклов равно n!, где n – это количество вершин графа. Таким образом, при увеличении количества вершин, задача быстро становится неразрешимой за разумное время для перебора вариантов.

Как правило, задача содержит дополнительное условие, что каждая вершина может быть пройдена только один раз – в таком случае, такой цикл будет называться гамильтоновым. В целях упрощения задачи и гарантии существования маршрута обычно считается, что граф задачи является полностью связным: между двумя вершинами обязательно есть ребро и только одно.

Задача называется симметричной, если стоимость перехода между двумя вершинами в одно и другое направление одинакова, то есть граф является неориентированным, и асимметричной, если стоимости переходов неравны, то есть граф ориентированный, а стоимость маршрута зависит также от направления обхода. В случае если граф задается матрицей смежности – матрицей, в которой каждой ячейке соответствует ребро между двумя вершинами с номерами равными номерам столбца и строки этой самой матрицы – для симметричной задачи коммивояжера матрица также будет симметрична относительно главной диагонали.

Задача имеет множество различных эффективных вариантов решений, которые сокращают полный перебор, все они являются эвристическими, это –

случайный перебор, жадные алгоритмы, имитация отжига, метод ветвей и границ, эволюционные и муравьиный алгоритмы.

1 История появления и развития

Муравьиный алгоритм основан на наблюдениях за муравьиными колониями. Суть наблюдений заключается в том, что при изучении новой территории муравьи изначально перемещаются случайно, почти хаотично. Разведывая местность и добывая пищу, муравей оставляет за собой след из феромонов, таким образом, как бы запоминая маршрут. Впоследствии он с большей вероятностью вернется на наиболее интересный ему маршрут, и другие муравьи с большей вероятностью пойдут по его следу. Таким образом, образуются феромоновые тропы – кратчайшие пути от колонии до источников пищи.

Муравьиные алгоритмы, как и многие эволюционные, основаны на популяциях потенциальных решений – поколениях искусственных муравьев. Первый вариант алгоритма был предложен Марко Дориго (1961) в 1991 году в его докторской диссертации.

В изначальном варианте вероятность перехода из -ой вершины в j-ую:

$$P_{ij} = \frac{{\tau_{ij}}^{\alpha} n_{ij}^{\beta}}{\sum {\tau_{ij}}^{\alpha} n_{ij}^{\beta}},$$

где τ_{ij} – количество феромона на ребре между вершинами;

n_{ij} – эвристическая информация, выгодность перехода между вершинами;

α и β– параметры.

Параметры α и β определяют баланс между предыдущим опытом маршрута и эвристической информацией. Эвристическая информация n_{ij} чаще всего определяется обратным расстоянием между вершинами $n_{ij} = \frac{1}{d_{ij}}$.

На каждой итерации цикла обхода графа муравьями происходит приращение феромона на каждом ребре на некоторую величину, зависящую от маршрутов муравьев на данной итерации:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta\tau_{ij}.$$

Итерации алгоритма также называются поколениями муравьев. В каждом поколении муравьи по очереди проходят граф. Обновление феромонов после каждого муравья в поколении называется локальным обновлением, а обновление после целого поколения муравьев – глобальным. Количество муравьев в поколении K, как и количество поколений T, являются важными параметрами алгоритма.

В целом, решение задачи муравьиный алгоритмом не является точным и даже может быть одним из худших, однако, при удачно подобранных параметрах, алгоритм обычно даёт близкий к оптимальному решению результат.

В настоящее время муравьиные алгоритмы получили применение при решении следующих практических задач [2]:

1) маршрутизация (прежде всего, в сетях – networkrooting);
2) задачи назначения (quadratic assignment problem, graph coloring, generalized assignment, frequentially assignment);
3) машинное обучение (classification rules, Bayesian networks, fuzzy systems);
4) кластеризация данных;
5) роботика (vehiclerouting и др.);
6) календарное планирование и составление расписания (job shop, open shop, flow shop, total terdiness, project sheduling, group shop);
7) покрытие множества, задача об укладке рюкзака (multi-knapsack, max independent set, reduncy allocation, set covering, maximum clique, weight constrained graph tree partition, bin packing);
8) биоинформатика;
9) обработка текстов.

2 Модификации алгоритма

Существует множество модификаций алгоритма. Представим некоторые из них, те, которые будут реализованы в нашей программе.

2.1.Обновление феромона М. Дориго

У М. Дориго существует три модификации (соответственно quantity, density и cycle):

$$\Delta\tau_{ij} = \frac{Q}{d_{ij}}, \qquad \Delta\tau_{ij} = Q, \qquad \Delta\tau_{ij} = \frac{1}{L^k}$$

где Q – запас феромона, некоторое константное значение,

L^k – длина маршрута -ого муравья.

2.2.Начало пути муравьев

Муравьи могут начинать движение все из одной случайной вершины, все из разных вершин или случайным образом. Важно на каждой итерации менять место начала движения, так будет меньше шансов, что алгоритм наткнется на локальный оптимум и не исследует большую часть графа. Распределение мест начала движения в рамках одной итерации имеет большее значение при использовании локального обновления феромона.

2.3. Высыхание феромона

Важной модификацией является добавление процесса высыхания феромона в графе. Во-первых, это уточнение приближает алгоритм к картине реального мира, во-вторых, позволяет алгоритму забыть неинтересные маршруты. Это дополнение позволит значительно увеличить количество поколений без ущерба качеству работы алгоритма из-за стремительного роста количества феромонов. Осуществляется высыхание по следующей формуле:

$$\tau(t+1) = (1-\rho)\tau(t),$$

где $\rho = [0:1]$ – скорость высыхания феромона.

2.4.Элитарные муравьи

Модификация заключается в добавлении к каждому поколению - количества муравьев, которые ходят и оставляют след только на лучшем маршруте этого поколения. Таким образом, алгоритм лучше запоминает хорошие маршруты.

3 Реализация и исследование алгоритма

Алгоритм был реализован на языке Python с использованием библиотеки numpy. Код алгоритма представлен в приложении. Исследование производили с использованием Jupyter Notebook.

Сгенерируем матрицу смежности и построим граф. Отрисовку графа производим с использованием библиотеки networkx.

```
[[ 0 44 29 13 46 16 38 14 26 49]
 [28  0 41 46 15 24 26  1 45 46]
 [12 16  0 10 38 16 19  2  9 48]
 [12 11 16  0 11 23 26 38 13 48]
 [ 1 41 12 17  0 38 11 27 37 29]
 [34 10 31 43 35  0 28 13 39 32]
 [38 26 43  1 23 48  0  3 36 41]
 [49 21  2 42 36 16  5  0 38 23]
 [45 29 15 26 34 10 36 25  0 47]
 [37 18  3 31 32  1 47  7  3  0]]
```

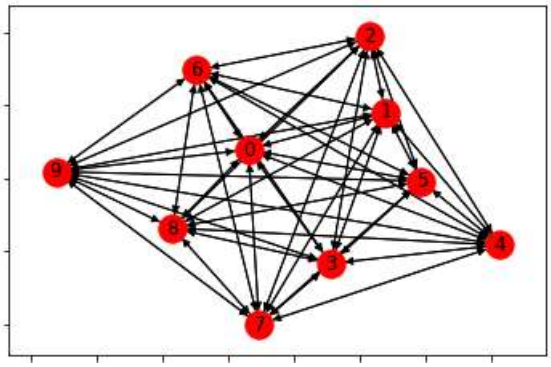

Рисунок 1 – Тестовый граф и его матрица смежности

Для графа с десятью вершинами использовали полный перебор при нахождения истинного решения задачи.

```
%%time
print(min(brute_force(L, [], 0)))
```

```
81
Wall time: 4.52 s
```

```
%%time
tao = aa.fit(L)
print(aa.get_best_dist())
```

```
C:\Users\mrnik\MyAntAlgorit
  rev_L = 1/L
```

```
93.0
Wall time: 1.02 s
```

Рисунок 2 – Сравнение результатов полного перебора (слева) и муравьиного алгоритма (справа)

Для графов большего размера полный перебор займет слишком много времени, так как время вычисления растет как факториал размера графа.

С использованием библиотеки seaborn построили тепловую карту распределения феромона на ребрах графа.

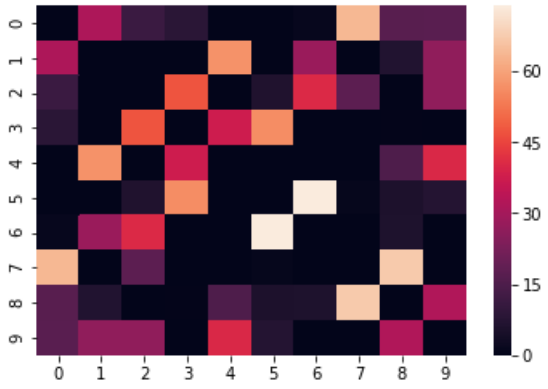

Рисунок 3 – Тепловая карта феромона на ребрах графа

При более правильном подборе параметров можно добиться приемлемых результатов в точности и во времени.

```
%%time
tao = aa.fit(L)
print(aa.get_best_dist())

86.0
Wall time: 1.06 s

%%time
print(min(brute_force(L, [], 0)))

85
Wall time: 4.95 s
```

Рисунок 4 – Сравнение результатов настроенного муравьиного алгоритма (сверху) и полного перебора (снизу)

Произведем подбор параметров алгоритма по сетке параметров, определим лучшие результаты на новом графе. Результаты подбора параметров α и β приведены в таблице 1, результаты подбора коэффициента высыхания ρ и количества элитных муравьев e приведены в таблице 2:

Таблица 1 – Результаты подбора параметров α и β

$\alpha \setminus \beta$	1	2	5	10
0.5	104	109	104	104
1	107	107	104	104
2	113	109	104	104

Таблица 2 – Результаты подбора параметров ρ и e

ρ \ e	2	5	10
0.5	110	109	117
0.7	107	107	114
0.9	118	107	127

По результатам работ делаем некоторый вывод. Так, наилучшие результаты были получены при (α, β) = (0.5, 1), (1, 5) (1, 10), (2, 10). Количество элитных муравьев следует выбирать как $e \leq 0.5\,K$, где K это число муравьев. Скорость высыхания ρ следует выбирать в соответствии с величиной запаса феромона Q и средней длиной дуги или пути, это зависит от выбора модификации обновления феромона, так, чтобы эти значения были сопоставимы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italie, – 1992.
2. Ю. Скобцов, Д. Сперанский, Эволюционные вычисления, лекция 12 муравьиные алгоритмы / Московский государственный университет путей сообщения : [эл. рес.] https://www.intuit.ru/studies/courses/14227/1284/lecture/24190
3. Цыпкин Я.З. Основы теории обучающихся систем. – М.:Наука. – 1970.
4. Скобцов Ю.А. Основы эволюционных вычислений. – Донецк: ДонНТУ, – 2008. – 326с.
5. С.Д. Штовба Муравьиные алгоритмы, ExponentaPro – 2003.
6. Ю. Зайченко, Н. Мурга, Исследование муравьиных алгоритмов оптимизации в задаче коммивояжера, International Journal "Information Models and Analyses" – 2013.

ПРИЛОЖЕНИЕ

```
import numpy as np

START_FROM = [
    'random',
    'form_one',
    'uniform'
]

ANT_SYS_MODIF = [
    'cycle',
    'density',
    'quantity'
]

class AntAlgorithm:

    def __init__(self,
                 a=1,  # коэффициент запаха
                 b=1,  # коэффициент расстояния
                 rho=0.5,  # коэффициент высыхания
                 Q=100,  # количество выпускаемого феромона
                 e=0,  # количество элитных муравьев
                 local_refresh=False,  # локальное обновление феромона
                 where_to_start='random',  # место начала движения муравьев
                 as_modif='density',  # модификация обновления феромона
                 random_seed=-1  # фиксация генератора случайных чисел
                 ):
        self.a = a
        self.b = b
        self.rho = rho
        self.Q = Q
        self.e = int(e)
        self.local_refresh = local_refresh
        if where_to_start in START_FROM:
            self.where_to_start = where_to_start
        else:
            raise ValueError('wrong where to start')
        if as_modif in ANT_SYS_MODIF:
            self.as_modif = as_modif
        else:
```

```
            raise ValueError('wrong ant-system modification name')
        if random_seed > 0:
            np.random.seed(random_seed)

    def fit(self,
            L,  # Матрица смежности графа
            AGES=-1,  # количество поколений
            ANTS=-1,  # количество муравьев в поколении
            ph=-1  # начальное значение ферромона

            ):
        self.L = L
        self.CITIES = len(L)
        if AGES > 0:
            self.AGES = int(AGES)
        else:
            self.AGES = self.CITIES * 50
        if ANTS > 0:
            self.ANTS = int(ANTS)
        else:
            self.ANTS = self.CITIES
        if ph >= 0:
            self.ph = ph
        else:
            self.ph = self.Q/self.CITIES

        # инициализация матрицы "краткости" дуг графа
        rev_L = 1/L
        # инициализация матрицы феромонов
        tao = np.ones((self.CITIES, self.CITIES)) * self.ph

        self.BEST_DIST = float("inf")                # лучшая длина маршрута
        self.BEST_ROUTE = None                        # лучший маршрут
        # матрица маршрутов муравьев в одном поколении (номера узлов графа)
        antROUTE = np.zeros((self.ANTS, self.CITIES))
        # вектор длины маршрута муравьев в одном поколении
        antDIST = np.zeros(self.ANTS)
        # вектор лучших длин маршрутов в каждом поколении
        self.antBEST_DIST = np.zeros(self.AGES)
        self.antAVERAGE_DIST = np.zeros(self.AGES)

        # основной цикл алгоритма
        # ---------- начало освновного цикла ----------
```

```
for age in range(self.AGES):
    antROUTE.fill(0)
    antDIST.fill(0)

    # ---------- начало цикла обхода графа муравьями ----------
    for k in range(self.ANTS):

        if self.where_to_start == 'random':
            # начальное расположение муравья в графе (случайное)
            antROUTE[k, 0] = np.random.randint(
                low=0, high=self.CITIES-1, size=1)
        elif self.where_to_start == 'uniform':
            # начальное расположение муравья в графе (равномерное)
            antROUTE[k, 0] = k % self.CITIES
        elif self.where_to_start == 'from_one':
            # начальное расположение муравья в графе (все с одного)
            antROUTE[k, 0] = 1
        else:
            assert 'wrong where to start'

        # ---------- начало обхода графа k-ым муравьем ----------
        for s in range(1, self.CITIES):
            # текущее положение муравья
            from_city = int(antROUTE[k, s-1])
            P = (tao[from_city] ** self.a) * \
                (rev_L[from_city] ** self.b)
            # вероятность посещения уже посещенных городов = 0
            for i in range(s):
                P[int(antROUTE[k, i])] = 0

            # вероятность выбора направления, сумма всех P = 1
            assert (np.sum(P) > 0), \
                "Division by zero. P = %s,"\
                " \n tao = %s \n rev_L = %s" % (
                P, tao[from_city], rev_L[from_city])
            P = P / np.sum(P)
            # выбираем направление
            isNotChosen = True
            while isNotChosen:
                rand = np.random.rand()
                for p, to in zip(P, list(range(self.CITIES))):
                    if p >= rand:
                        # записываем город №s в вектор k-ого муравья
```

```
                    antROUTE[k, s] = to
                    isNotChosen = False
                    break
            # локальное обновление феромона
            if self.local_refresh:
                for s in range(self.CITIES):
                    city_to = int(antROUTE[k, s])
                    city_from = int(antROUTE[k, s-1])
                    if self.as_modif == 'cycle':
                        tao[city_from, city_to] = \
                            tao[city_from, city_to] + \
                            (self.Q / antDIST[k])
                    elif self.as_modif == 'density':
                        tao[city_from, city_to] = tao[city_from,
                                                      city_to] + self.Q
                    elif self.as_modif == 'quantity':
                        tao[city_from, city_to] = \
                            tao[city_from, city_to] + \
                            (self.Q / L[city_from, city_to])
                    else:
                        assert 'wrong ant-system modification name'
                    tao[city_to, city_from] = tao[city_from,
                                                  city_to]
        # ---------- конец цикла обхода графа ----------

        # вычисляем длину маршрута k-ого муравья
        for i in range(self.CITIES):
            city_from = int(antROUTE[k, i-1])
            city_to = int(antROUTE[k, i])
            antDIST[k] += L[city_from, city_to]

        # сравниваем длину маршрута с лучшим показателем
        if antDIST[k] < self.BEST_DIST:
            self.BEST_DIST = antDIST[k]
            self.BEST_ROUTE = antROUTE[k]
    # ---------- конец цикла обхода графа муравьями ----------

    # ---------- обновление феромонов----------
    # высыхание по всем маршрутам (дугам графа)
    tao *= (1-self.rho)

    # цикл обновления феромона
    for k in range(self.ANTS):
```

```
    for s in range(self.CITIES):
        city_to = int(antROUTE[k, s])
        city_from = int(antROUTE[k, s-1])
        if self.as_modif == 'cycle':
            tao[city_from, city_to] = \
                tao[city_from, city_to] + \
                (self.Q / antDIST[k])
        elif self.as_modif == 'density':
            tao[city_from, city_to] = \
                tao[city_from, city_to] + self.Q
        elif self.as_modif == 'quantity':
            tao[city_from, city_to] = \
                tao[city_from, city_to] + \
                (self.Q / L[city_from, city_to])
        else:
            assert 'wrong ant-system modification name'
        tao[city_to, city_from] = tao[city_from, city_to]

# проход элитных e-муравьев по лучшему маршруту
if self.e > 0:
    for s in range(CITIES):
        city_to = int(BEST_ROUTE[s])
        city_from = int(BEST_ROUTE[s-1])
        if self.as_modif == 'cycle':
            tao[city_from, city_to] = \
                tao[city_from, city_to] + \
                ((self.Q * self.e) / self.BEST_DIST)
        elif self.as_modif == 'density':
            tao[city_from, city_to] = \
                tao[city_from, city_to] + self.Q * self.e
        elif self.as_modif == 'quantity':
            tao[city_from, city_to] = \
                tao[city_from, city_to] + \
                ((self.Q * self.e) / L[city_from, city_to])
        else:
            assert 'wrong ant-system modification name'
        tao[city_to, city_from] = tao[city_from, city_to]

# ---------- конец обновления феромона ----------

# конец поколения муравьев

# сбор информации для графиков
```

```
            self.antBEST_DIST[age] = self.BEST_DIST
            self.antAVERAGE_DIST[age] = np.average(antDIST)

        return tao

    def get_best_route(self):
        return self.BEST_ROUTE

    def get_best_dist(self):
        return self.BEST_DIST

    def get_best_dists(self):
        return [age for age in range(self.AGES)], self.antBEST_DIST

    def get_average_dists(self):
        return [age for age in range(self.AGES)], self.antAVERAGE_DIST
```