Міністерство освіти та науки України
Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Звіт до лабораторної роботи №3 на тему: "Мурашиний алгоритм"

Виконав студент групи ОМ-3
Скибицький Нікіта

Київ, 2019

Зміст

1 Постановка задачі **2**

2 Опис алгоритму **3**
- 2.1 Феромени які залишає мурашка 3
- 2.2 Процес побудови розв'язку 3
- 2.3 Процес оновлення феромен 4

3 Код **4**
- 3.1 Процес побудови розв'язку 4
- 3.2 Процес обчислення феромен 5
- 3.3 Програма-драйвер і основний алгоритм 5

4 Результати **6**

1 Постановка задачі

Розглянемо відому задачу про ранець: є T типів предметів, причому предметів i-го типу рівно q_i штук. Предмет i-го типу характеризується значеннями своєї корисності u_i та ваги w_i. Окрім цього є ранець який вміщує довільну кількість предметів сумарною вагою не більше W. Необхідно вибрати підмножину заданих предметів яку можна розмістити у ранці, з максимальною сумарною корисністю.

Тобто, ставиться задача

$$U(c) = \sum_{i=1}^{T} c_i \cdot u_i \xrightarrow[c \in \mathcal{C}]{} \max, \tag{1.1}$$

де $\mathcal{C}$ — допустима область, яка визначається наступним чином:

$$\mathcal{C} = \left\{ c \in \mathbb{Z}^T \middle| \forall i : 0 \le c_i \le q_i \land \sum_{i=1}^{T} c_i \cdot w_i \le W \right\}. \tag{1.2}$$

Зауважимо, що можна також записати

$$U(c) = \langle c, u \rangle, \tag{1.3}$$

і

$$\mathcal{C} = \{ c \in Q_1 \times Q_2 \times \ldots \times Q_T | \langle c, w \rangle \le W \}, \tag{1.4}$$

де $Q_i = \mathbb{Z} \cap [0, q_i]$.

2 Опис алгоритму

Розглянемо популяцію з N мурах, які протягом M ітерацій намагаються спакувати свій “ранець” (наприклад, підготувати запаси на зиму, які необхідно розмістити в обмеженому просторі мурашника). Уявімо, що на кожній ітерації кожна мурашка пакує свій уявний ранець ходячи туди-сюди до потрібних їй предметів, залишаючи на своєму шляху феромени, і керуючись у вже наявними із попередніх ітерацій феромонами для вибору шляху.

2.1 Феромени які залишає мурашка

А саме, нехай муршака знайшла розв’язок c з сумарною вагою w і сумарною корисністю u, тоді кількість феромонів f_i залишених на шляху до купки об’єктів типу i описується функцією f:

$$f_i = f(c_i, w, u, w_i, u_i), \tag{2.1}$$

на яку накладаються наступні умови:

$$\frac{\partial f}{\partial w} < 0, \tag{2.2}$$

$$\frac{\partial f}{\partial u} > 0, \tag{2.3}$$

$$\frac{\partial f}{\partial w_i} < 0, \tag{2.4}$$

$$\frac{\partial f}{\partial u_i} > 0, \tag{2.5}$$

$$\frac{\partial f}{\partial c_i} > 0, \tag{2.6}$$

$$f(0, \ldots) = 0. \tag{2.7}$$

Наприклад,

$$f(w, u, w_i, u_i) = \ln(c_i + 1) \cdot \frac{u_i}{w_i} \cdot \frac{u}{w}. \tag{2.8}$$

2.2 Процес побудови розв’язку

Нехай з попередньої ітерації вже наявні феромени у кількості f_i на шляху до купки предметів i-го типу, тоді поки у ранець можна покласти ще хоча б один предмет мурашка обчислює наступні характеристики χ_i кожного типу предметів який ще може влізти у ранець:

$$\chi_i = \chi(u_i, w_i, f_i), \tag{2.1}$$

де на функцію χ накладаються наступні умови

$$\frac{\partial \chi}{\partial w_i} < 0, \tag{2.2}$$

$$\frac{\partial \chi}{\partial u_i} > 0, \tag{2.3}$$

$$\frac{\partial \chi}{\partial f_i} > 0. \tag{2.4}$$

Наприклад,

$$\chi(w_i, u_i, f_i) = \frac{u_i}{w_i} \cdot (1 + \rho \cdot f_i), \tag{2.5}$$

де $\rho \in (0, 2)$ — певний коефіцієнт, наприклад $\rho = 1$.

На основі обчислених характеристик обчислюються ймовірності вибору кожного типу предметів:

$$p_i = \frac{\chi_i}{\sum\limits_{i=1}^{T} \chi_i}. \tag{2.6}$$

2.3 Процес оновлення фероменів

Якщо до ітерації i кількість фероменів на шляху до j-ої купки предметів дорівнює $F_{i,j}$, а на i-ій ітерації додалося $f_{i,j}$, то

$$F_{i+1,j} = \alpha \cdot f_{i,j} + (1 - \alpha) \cdot F_{i,j}, \tag{2.1}$$

де $\alpha \in (0, 1)$ — певний коефіцієнт, наприклад $\alpha = 0.1$.

3 Код

3.1 Процес побудови розв'язку

```python
#!/usr/bin/env python
import numpy as np
from typing import Tuple


def chi(w: np.array, u: np.array, f: np.array,
        rho: float=1.) -> np.array:
    return (u / w) * (1 + rho * f)
```

```python
def generate_solution(T: int, q: np.array, w: np.array, u: np.array,
                W: int, F: np.array) -> Tuple[np.array, int, int]:
        c, _w, _u = np.repeat(0, T), 0, 0
        while True:
                _chi = chi(w, u, F) * (w <= W) * (q > c)
                if sum(_chi) == 0:
                        return c, _w, _u
                _chi /= sum(_chi)
                i = np.random.choice(np.arange(T), p=_chi)
                W -= w[i]
                _w += w[i]
                _u += u[i]
                c[i] += 1
```

3.2 Процес обчислення фероменів

```python
#!/usr/bin/env python
import numpy as np

def calculate_feroments(c: np.array, _w: float, _u: float,
                w: np.array, u: np.array) -> np.array:
        f = np.log(c + 1) * (u / w) * (_u / _w)
        return np.array(f, dtype='float64')
```

3.3 Програма-драйвер і основний алгоритм

```python
#!/usr/bin/env python
import numpy as np
from generate_solution import generate_solution
from calculate_feroments import calculate_feroments

np.set_printoptions(precision=3)

np.random.seed(0)

T = 100
q = np.random.choice(np.arange(1, 5), size=T)
w = np.random.choice(np.arange(10, 20), size=T)
u = np.random.choice(np.arange(100, 200), size=T)
W = 1_000
```

```
N = 100
M = 100
alpha = .1
F = np.array(np.repeat(100, T), dtype='float64')
best_u = 0

for i in range(M):
        f = np.array(np.repeat(0, T), dtype='float64')
        for j in range(N):
                c, _w, _u = generate_solution(T, q, w, u, W, F)
                best_u = max(_u, best_u)
                f += calculate_feroments(c, _w, _u, w, u)
        print(f'it #{i}: best_u = {best_u}')
        F = alpha * f + (1 - alpha) * F

print(generate_solution(T, q, w, u, W, F))
print(u/w)
print(F)
```

4 Результати

Проводилося тестування на відносно складній задачі ($T = 100$) але однорідній задачі. Однорідність означає, що

$$\frac{\max_i u_i}{\min u_i} \leq 2 \quad \wedge \quad \frac{\max_i w_i}{\min w_i} \leq 2 \tag{4.1}$$

Початковий результат отриманий випадковим чином складав 80% теоретичного максимуму:

```
it #00: best_u = 11882
```

За 10 ітерацій алгоритм досяг покращення у 17% у порівнянні з початковим результатом, тобто досяг результату у 97% від теоретичного максимуму:

```
it #10: best_u = 13942
```

Опісля відбувається стагнація: за наступні 100 ітерацій прогрес склав ще 2%, у результаті чого алгоритм підійшов до 99% теоретичного максимуму.

```
it #99: best_u = 14135
```