`ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Системы искусственного интеллекта»

ОТЧЁТ
ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №9
«ИНС Хопфилда. Задача о назначениях»

Выполнили студенты — Шефнер А.
Факультет: АИТ — Кот. Н.Д.
Группа: ИВБ-211 — Егупов Н.М.
Ахмедов Х.А.

Проверил: Пугачев С.В.

Санкт-Петербург

2025

Задание

Задача о назначениях (Assignment Problem) — классическая задача оптимизации, возникающая в экономике и логистике.

Формулировка

Дано:

- Дано множество испольнителей $I = \{1, 2, \ldots, n\}$ и множество работ $J = \{1, 2, \ldots, n\}$
- Каждый исполнитель может быть назначен только на одну работу, и каждая работа может выполняться только одним исполнителем.
- Матрица стоимостей: c_{ij} (стоимость выполнения работы j исполнителем i)
- Для каждой пары «исполнитель-работа» задана стоимость выполнения работы c_{ij}, где $i \in I$, $j \in J$.

По условию задачи требуется найти такое распределение работ по исполнителям, чтобы общая стоимость была минимальной:

Минимизировать: $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$, где x_{ij} - переменная, равная 1,если исполнитель i назначен на работу j и 0 в противном случае

Ход работы

Ограничения

1. Каждый исполнитель назначается на одну работу:

$$\sum_{j=1,i=1}^{n} x_{ij} = 1 \quad \forall i, j \in I, J$$

Разработка класса сети Хопфилда для решения задачи назначения

Атрибуты класса:

- `cost_matrix` : матрица стоимостей (n x n)
- `A` : коэффициент для ограничения по строкам
- `B` : коэффициент для ограничения по столбцам
- `C` : коэффициент для минимизации стоимости

- `max_iter` : максимальное число итераций
- `T_start` : начальная температура для имитации отжига
- `cooling_rate` : скорость охлаждения

In [1]:
```python
import numpy as np

class HopfieldNetwork:
    def __init__(self, cost_matrix, A=1.0, B=1.0, C=1.0, max_iter=1000, T
        self.n = len(cost_matrix)
        self.cost_matrix = np.array(cost_matrix, dtype=float)
        self.A = A
        self.B = B
        self.C = C
        self.max_iter = max_iter
        self.T_start = T_start
        self.cooling_rate = cooling_rate
        # Нормализация начальных значений по строкам
        self.V = np.random.rand(self.n, self.n)
        self.V /= np.sum(self.V, axis=1, keepdims=True)

    def update(self, T):
        # Инициализация матрицы состояний нейронов V (n x n)
        # Каждый нейрон V[i,j] представляет вероятность назначения i-го р
        row_sums = np.sum(self.V, axis=1) - 1
        col_sums = np.sum(self.V, axis=0) - 1
        # Асинхронное обновление в случайном порядке
        indices = [(i, j) for i in range(self.n) for j in range(self.n)]
        np.random.shuffle(indices)
        for i, j in indices:
             # Вычисление общего входа нейрона (i,j):
            # 1. Штраф за нарушение ограничения по строке
            # 2. Штраф за нарушение ограничения по столбцу
            # 3. Вклад от стоимости назначения
            u_ij = (-self.A * row_sums[i]
                    - self.B * col_sums[j]
                    - self.C * self.cost_matrix[i, j])
            # Ограничение для предотвращения переполнения
            u_ij_clipped = np.clip(u_ij, -700 * T, 700 * T)
            self.V[i, j] = 1 / (1 + np.exp(-u_ij_clipped / T))

    def anneal(self):
        T = self.T_start
        for _ in range(self.max_iter):
            self.update(T)
            T *= self.cooling_rate  # Понижение температуры

    def get_solution(self):
        # Преобразование непрерывных состояний нейронов в дискретное реше
        # Возвращает:
        # - Матрицу назначений (0 и 1), где 1 означает назначение
        solution = np.zeros((self.n, self.n), dtype=int)
        V_rounded = np.round(self.V).astype(int)

        # Коррекция строк
        for i in range(self.n):
            row = V_rounded[i, :]
            if np.sum(row) == 0:
                max_j = np.argmax(self.V[i, :])
                V_rounded[i, max_j] = 1
```

```
            elif np.sum(row) > 1:
                max_j = np.argmax(self.V[i, :])
                V_rounded[i, :] = 0
                V_rounded[i, max_j] = 1

        # Коррекция столбцов
        for j in range(self.n):
            col = V_rounded[:, j]
            if np.sum(col) == 0:
                available_rows = np.where(np.sum(V_rounded, axis=1) == 0)
                if len(available_rows) > 0:
                    min_row = min(available_rows, key=lambda x: self.cost
                else:
                    min_row = np.argmin(self.cost_matrix[:, j])
                V_rounded[min_row, j] = 1
            elif np.sum(col) > 1:
                candidates = np.where(col == 1)[0]
                min_row = min(candidates, key=lambda x: self.cost_matrix[
                V_rounded[:, j] = 0
                V_rounded[min_row, j] = 1

        return V_rounded
```

Проверка

3 работника с 3 видами работ

In [2]:
```
# Пример использования
cost_matrix = np.array([
    [2, 3, 1],
    [5, 1, 4],
    [3, 2, 2]
])
print("Матрица стоимостей:")
display(cost_matrix)

hn = HopfieldNetwork(
    cost_matrix,
    A=5.0,
    B=5.0,
    C=1.0,
    max_iter=1000,
    T_start=100.0,
    cooling_rate=0.99
)
hn.anneal()
solution = hn.get_solution()

print("Матрица назначений:")
display(solution)
print("Суммарная стоимость:", np.sum(solution * cost_matrix))
```

```
Матрица стоимостей:
array([[2, 3, 1],
       [5, 1, 4],
       [3, 2, 2]])
Матрица назначений:
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
Суммарная стоимость: 5
```

Вывод

В ходе данной лабораторной работы были отработаны практические навыки моделирования и реализации нейронной сети Хопфилда и применение ее в решении задачи о назначениях.