ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

Phương pháp thiết kế thuật toán gần đúng

Nhóm 5 thực hiện

Môn học: CS112.P11.KHTN

Sinh viên thực hiện:

Nguyễn Thiên Bảo - 23520127

Trần Lê Minh Nhật - 23521098

Giáo viên hướng dẫn:

Nguyễn Thanh Sơn

ngày 20 tháng 10 năm 2024

1 Bài toán Set Cover

1.1 Mô tả bài toán

- **Đầu vào:**
  - Tập $U = \{u_1, u_2, \ldots, u_n\}$: tập hợp các phần tử cần được bao phủ.
  - Tập $S = \{S_1, S_2, \ldots, S_m\}$: tập các tập con của U, trong đó mỗi tập $S_i \subseteq U$.
- **Đầu ra:**
  - Một tập $S' \subseteq S$, sao cho $\bigcup_{S_i \in S'} S_i = U$ và $|S'|$ là nhỏ nhất.

Ví dụ minh họa:

- Tập U: $\{1, 2, 3, 4, 5\}$.
- Tập S: $S_1 = \{1, 2\}, S_2 = \{2, 3\}, S_3 = \{3, 4, 5\}$.
- Lời giải tối ưu: $S' = \{S_1, S_3\}$.

1.2 Cách giải

1.2.1 Phương pháp 1: Thuật toán tham lam

- **Ý tưởng:**
  1. Bắt đầu với tập rỗng S'.
  2. Mỗi bước, chọn tập $S_i \in S$ sao cho nó bao phủ được nhiều phần tử chưa được bao phủ nhất trong U.
  3. Cập nhật tập S', đồng thời loại bỏ các phần tử đã được S_i bao phủ khỏi U.
  4. Lặp lại cho đến khi U rỗng.
- **Độ phức tạp:** $O(mn)$, với m là số tập con và n là số phần tử trong U.

Code:

```
def greedy_set_cover(U, S):
    U = set(U)
    S = [set(s) for s in S]
    selected_subsets = []

    while U:
        best_subset = max(S, key=lambda s: len(s & U))
        selected_subsets.append(best_subset)
        U -= best_subset

    return selected_subsets
```

Listing 1: Thuật toán tham lam cho bài toán Set Cover

1.2.2 Phương pháp 2: Giải pháp lập trình phi tuyến tính

- **Ý tưởng:**

  1. Mô hình hóa bài toán dưới dạng chương trình tuyến tính nhị phân:

  $$\min \sum_{i=1}^{m} x_i$$

  với ràng buộc:

  $$\sum_{i:u_j \in S_i} x_i \geq 1, \quad \forall u_j \in U.$$

  2. Giải bài toán bằng cách thư giãn các ràng buộc $x_i \in \{0, 1\}$ thành $x_i \in [0, 1]$.

  3. Làm tròn nghiệm để thu được lời giải nguyên.

Code:

```
from scipy.optimize import linprog

def linear_programming_set_cover(U, S):
```

```
    n = len(U)
    m = len(S)
    c = [1] * m
    A = [[1 if u in S[i] else 0 for i in range(m)] for u in U]
    b = [1] * n

    result = linprog(c, A_ub=-np.array(A), b_ub=-np.array(b), bounds=(0, 1),
method='highs')
    return result.x
```

Listing 2: Giải pháp lập trình phi tuyến tính cho bài toán Set Cover

2 Bài toán TSP (Travelling Salesman Problem)

2.1 Mô tả bài toán

- **Đầu vào:**
  - Đồ thị $G = (V, E)$ với:
    - V: tập đỉnh đại diện các thành phố.
    - E: tập cạnh với trọng số là chi phí hoặc quãng đường giữa các đỉnh.
- **Đầu ra:** Một chu trình Hamilton có tổng trọng số nhỏ nhất.

Ví dụ minh họa:

- Tập đỉnh V: $\{A, B, C, D\}$.
- Trọng số cạnh:
  - $c(A, B) = 10$, $c(A, C) = 15$, $c(A, D) = 20$,
  - $c(B, C) = 35$, $c(B, D) = 25$, $c(C, D) = 30$.
- Lời giải tối ưu: Chu trình $A \to B \to D \to C \to A$ với chi phí 80.

2.2 Cách giải

2.2.1 Phương pháp 1: Thuật toán tham lam

- **Ý tưởng:**
  1. Bắt đầu từ một đỉnh bất kỳ.
  2. Ở mỗi bước, chọn cạnh có trọng số nhỏ nhất nối với đỉnh chưa được thăm.
  3. Tiếp tục đến khi quay về đỉnh ban đầu.
- **Độ phức tạp:** $O(n^2)$, với n là số đỉnh.

Code:

```
import numpy as np

def tsp_greedy(cost_matrix):
    n = len(cost_matrix)
    visited = [False] * n
    path = [0]
    visited[0] = True
    total_cost = 0

    for _ in range(n - 1):
        last = path[-1]
        next_city = np.argmin([cost_matrix[last][j] if not visited[j] else float('inf') for j in range(n)])
        total_cost += cost_matrix[last][next_city]
        path.append(next_city)
        visited[next_city] = True

    total_cost += cost_matrix[path[-1]][path[0]]
    path.append(path[0])

```

```
    return path, total_cost
```

Listing 3: Thuật toán tham lam cho bài toán TSP

2.2.2 Phương pháp 2: Heuristic Nearest Neighbor (NNH)

- **Ý tưởng:**
  1. Chọn đỉnh gần nhất ở mỗi bước.
  2. Tiếp tục đến khi tất cả các đỉnh được thăm.
- **Độ phức tạp:** $O(n^2)$.

Code:

```
def tsp_nearest_neighbor(cost_matrix):
    n = len(cost_matrix)
    visited = [False] * n
    path = [0]
    visited[0] = True
    total_cost = 0

    for _ in range(n - 1):
        last = path[-1]
        min_distance = float('inf')
        next_city = -1
        for j in range(n):
            if not visited[j] and cost_matrix[last][j] < min_distance:
                min_distance = cost_matrix[last][j]
                next_city = j
        total_cost += min_distance
        path.append(next_city)
        visited[next_city] = True

    total_cost += cost_matrix[path[-1]][path[0]]
```

```
21      path.append(path[0])
22
23      return path, total_cost
```

Listing 4: Heuristic Nearest Neighbor cho bài toán TSP