TSP via Dynamic Programming

Hongda Li

UBCO

December 1, 2022

ToC

Introduction

Define $G = (V, E)$ to be an undirected graph.

1. Dynamic programming is the idea of combining optimal solutions for smaller problems to make the full solution.
2. For the traveling salesman, we consider using spanning paths for covering different sizes of subsets.

Definition

Let $S \subseteq V$ and use $C(S, i, j)$ to denote the optimal $i \to j$ spaning path and its cost covering all vertices in S.

The algorithm

TSP using Dynamic Programming

Algorithm 1 Held Karp algorithm for Travelling Salesman

for $i, j \in V, i < j$ **do**
  C({i, j}, i, j) := c(i, j)
end for
for $k = 3, 4, \cdots n$ **do**
  for $|S| = k, S \subseteq V$ **do**
    for $i, j \in S, i < j$ **do**
      $C(S, i, j) := \min_{l \in S \setminus \{i,j\}} \{C(S \setminus \{j\}, i, l) + c(l, j)\}$
    end for
  end for
end for
return $\min_{i,j \in V} \{C(V, i, j) + c(i, j)\}$

Facts about the algorithm

1. Its complexity is $\mathcal{O}(n^3 2^n)$.
2. We need to keep track of the optimal solutions and the cost of the optimal solutions during the iterations of the algorithm.
3. Because this dynamic programming is a bottom-up approach, storing the results from previous iterations suffices. We used this strategy for our implementations.

Challenges and their solutions

1. Generating all subsets S with size k from V.
2. Choosing the data structure for $C(S, i, j)$.
3. Writing it out during the final few weeks of a term in grad school when time management is crucial.
4. The perfectionist tendency.

Challenges and their solutions

1. Generating all subsets S with size k from V.
2. Choosing the data structure for $C(S, i, j)$.
3. Writing it out during the final few weeks of a term in grad school when time management is crucial.
4. The perfectionist tendency.

Solution

Solution: Use python and pythonic code.

k-subsets

This is the inner function of another function that generates subsets given an array. It is a recursive yield function.

```python
def inner_recur(a, k, start_at, already_chosen):
    if k == 0:
        yield list(already_chosen)
        return
    n = len(a)
    for I in range(start_at, n - k + 1):
        already_chosen.append(a[I])
        yield from inner_recur(a, k - 1, I + 1, already_chosen)
        already_chosen.pop()
    return
```

Python Itertools

When we first wrote it, we were unaware that python "itertools" package exists natively. The package provides generators for subsets and permutations of a set.

Link is here. Let's explore it together.

Storing the subsets and i, j

1. Python's dictionary supports multiple arguments.
2. However, the argument has to be hashable.
3. We use a sorted "tuple" as the key for C to represent the subset $S \subseteq V$.

Inside the inner functions for carrying out the main algorithm, I use the following functions to index my set:

```python
def has_edge(i, j):
    return tuple(sorted([i, j])) in this.c

def C(S, i, j): # recursion table.
    return this.ctable[S, tuple(sorted([i, j]))]

def P(S, i, j): # get path
    return this.ptable[S, tuple([i, j])]

def edge_cost(i, j):
    return this.c[tuple(sorted([i, j]))]
```

google collab, the repo

- The repo containing our code is here
- We present our results for TSP via dynamic programming with my google collab notebook here