The Chinese University of Hong Kong, Shenzhen

CSC4120

Design and Analysis of Algorithms

Party Together Problem

Author 1:	Kangqi Yu
Student ID:	121090735
Author 2:	Yuzhe Yang
Student ID:	121090684
Author 3:	Haoqi Zhang
Student ID:	121090766

May 19, 2024

1 PTP Approaches

The pseudo-code of our solution will illustrated in Algorithm 1.

1.1 Environment

To install these dependencies:

```
pip install ortools
pip install networkx
```

1.2 Implementation

Initialization

We start by initializing the tour with the starting point, which is our house located at node 0. The pick-up locations dictionary such that each friend's pick-up location is initially set to their home. The initial tour T includes node 0 followed by all pick-up locations and ends at node 0. Then we compute the shortest paths between all pairs of nodes using the Floyd-Warshall algorithm. This dictionary is essential for quickly evaluating potential routes and costs.
In order to find the optimal solution, we define a function `compute_cost` that calculates both the driving cost and walking cost of a given tour and pick-up locations dictionary.

Iteration

We iteratively improve the tour and pick-up locations by exploring potential reassignments of friends to different pick-up locations. A priority queue is used to evaluate the cost of assigning each friend to each candidate pick-up location. In each iteration, the priority queue will pop the friend and candidate pick-up location with the lowest cost, and we will reassign the friend to that location if it results in a lower total cost.
Once the new pick-up locations are updated, we solve a TSP on the pick-up locations to determine the optimal tour using `ortools`.

TSP Solver

The solver begins by creating a routing index manager to map node indices to routing variable indices. Subsequently, a routing model is instantiated, and a `distance_callback` function is registered to compute distances between nodes. Arc costs are then defined, and a heuristic strategy is set for the optimization process. The solver returns a list containing the optimal path, representing the optimal solution to the TSP. After that, we will re-compute the cost of this tour. If it is a better solution, the tour T and pick-up location will be updated, and the algorithm proceeds to the next iteration.

Final Tour Adjustment

To ensure the final tour only includes existing edges in the graph, we adjust the tour by replacing any non-existent edges with the shortest path between the respective nodes. This guarantees that the tour adheres to the constraints of the problem.

2 Theoretical Questions

2.1 Question 5.1

To show that PTP is NP-hard, we can find values for α for which PHP = PTP, since 0 $\leq \alpha \leq 1$, we can use $\alpha = 0$, when $\alpha = 0$, that means any walking cost will result in a total cost that larger than 0. So we need to make the total cost consist of driving costs, which will finally cost 0. So when α equals 0, PHP = PTP, which means the solution of PHP is obtained by solving PTP. Since PHP is known as NP-hard, then PTP is also NP-hard.

2.2 Question 5.2

First, we will show that the cost of PHP is at most twice that of the optimal solution. Just think of an optimal solution as $C_{ptpopt} = \alpha \sum_{i=1}^{n} w_{u_{i-1}u_i} + \sum_{m=0}^{|F|-1} d_{p_m h_m}$. We can let all our friends not walk and stay home. The cost will be $C_0 = \alpha \sum_{i=1}^{n} w_{u_{i-1}u_i} + 2\alpha \sum_{m=0}^{|F|-1} d_{p_m h_m}$, which is less or equal (equality holds when $\alpha = 1$ and $\sum_{i=1}^{n} w_{u_{i-1}u_i} = 0$) than 2 times of the optimal from PTP (for α is in (0, 1) and all weights are positive). As the optimal of PHP is less than the above cost (for the above cost is one of the candidates in PHP), $C_{php} \leq C_0 \leq 2C_{ptpopt}$. So, the cost of PHP is at most twice that of the optimal solution. The equality holds when there is only one friend; it is connected with the source, and $\alpha = 1$, which means the bound is tight.

Appendix: PTP Solver Algorithm

Algorithm 1 PTP Solver Algorithm

Initialize tour T with starting point 0
Compute shortest paths between all nodes, store in D
Initialize pick-up locations dictionary P with each friend's home as their pick-up location
Create initial tour T including all pick-up locations and returning to start
improvement $\leftarrow$ True
while *improvement* == True **do**
    Compute the current best cost: c
    Create an empty priority queue q
    for each friend f **do**
        for each candidate pick-up location $f_{\text{candidate}}$ **do**
            Compute cost $c(f \rightarrow f_{\text{candidate}})$ of assigning f to $f_{\text{candidate}}$
            Push $(c(f \rightarrow f_{\text{candidate}}), f, f_{\text{candidate}})$ to q
        end for
    end for
    while q is not empty **do**
        Pop the assignment $(c_{\min}, f_{\min}, f_{\text{candidate}})$ from q
        Create a new pick-up locations dictionary P_{new} based on $f_{\min}$
        Create a new tour T_{new} based on P_{new}
        $T_{\text{new}} \leftarrow$ TSP_Solver(T) {Solve TSP for the new tour}
        Compute the new cost: c_{new}
        if $c_{\text{new}} \leq c$ **then**
            Update best cost, tour, and pick-up locations
            Set *improvement* $\leftarrow$ True
            break
        end if
    end while
    Set *improvement* $\leftarrow$ False {If no improvement found in this iteration}
end while