CSC 4120 Project
Party Together

November 9, 2025 Version 3.0

1 Problem Statement

You and your friends are going to throw a party at your house this weekend to celebrate the end of the semester. You, being an excellent driver with a nice car, offer to pick up all your friends near or at their homes and drive them to your house. Can you come up with a transportation plan so that everyone gets to the party as efficiently as possible?

Formally, you are given an instance of the Party Together Problem (PTP) with inputs (G, H, α). $G = (V, E)$ is an undirected graph where V is the set of nodes indexed from 0 to $|V| - 1$ and each node represents a location in the city. The weight of each edge (u, v) is the length of the direct road connection between location u and v, which is non-negative. Your house is at location 0. $H \subset V$ is the set of distinct locations that correspond to your friends' homes. If $F = \{0, 1, \dots, |F| - 1\}$ is the set of friends, then each friend $m \in F$ has a house location $h_m \in H$ and each house location corresponds to exactly one friend. The constant α refers to the relative cost of driving vs walking.

A possible pickup schedule specifies the locations where you will pick up your friends. More specifically, it will specify for each friend m a pickup location $p_m \in V$, and you will need to drive your car starting from your home to pass from all the pickup locations to collect your friends and bring them back home (assume that the car has enough capacity to carry all your friends). However, your friends can not walk too far from their homes. We therefore restrict that you can only pick up each friend either at their home or at one of the neighboring locations of their home. That is to say, for all m

$$p_m \in \{h_m\} \cup \mathcal{N}(h_m)$$

where $\mathcal{N}(h_m)$ is the set of neighboring nodes of h_m in the graph.

Cost structure: Each friend incurs a walking cost equal to the distance traveled to get from his home to his pickup location. You incur a driving cost equal to the distance traveled by your car multiplied by the constant $\alpha, 0 \leq \alpha \leq 1$. It is in general more efficient to travel by car than walking, and the cost of the car does not depend on how many people it carries.

Assumptions:

- The capacity of your car is unlimited.
- Your friend would take direct edge from her home to her pick-up location or stay at home if pick-up location is her home.
- You can pick up multiple friends at the same location.

- You may pass from the same location more than once.
- The graph is connected.
- There is no road from a location to itself, i.e., there is no edge (i, i) in the graph for any i.
- Triangle inequality holds. Taking the direct path between two locations (if it exists) is always not longer than going through some other intermediate location, i.e., for any edge (i, j) in the graph and any location $u \neq i, j$,

$$w_{ij} \leq d_{iu} + d_{uj},$$

where d_{xy} is the length of the shortest path from location x to location y.

Your task is to find i) the set of pick-up locations for your friends, and ii) a routing schedule of the car, i.e., a 'tour'[1] that includes node 0 and the above pickup locations, that minimize the total cost.

The total cost is calculated as follows. Let $L = \{p_m\}_{m \in F}$ be the set of locations you pick up your friends, and $\{u_0, u_1, \cdots, u_n\}$, $u_0 = u_n = 0$, be the tour of the car where $L \subseteq \{u_0, \ldots, u_{n-1}\}$. Let d_{ij} be the length of the shortest path between locations i and j (define $d_{ii} = 0$). The total cost corresponding to this tour and set of pickup locations is

$$\alpha \sum_{i=1}^{n} w_{u_{i-1}u_i} + \sum_{m=0}^{|F|-1} d_{p_m h_m}.$$

For the **example** in Figure 1 where $H = \{1, 2, 3\}$ and $\alpha = \frac{2}{3}$, the optimal car tour is $0 \to 1 \to 0$ and pick up everyone at node 1. The total cost is $(2 * \frac{2}{3} + 1 + 1 = \frac{10}{3})$.

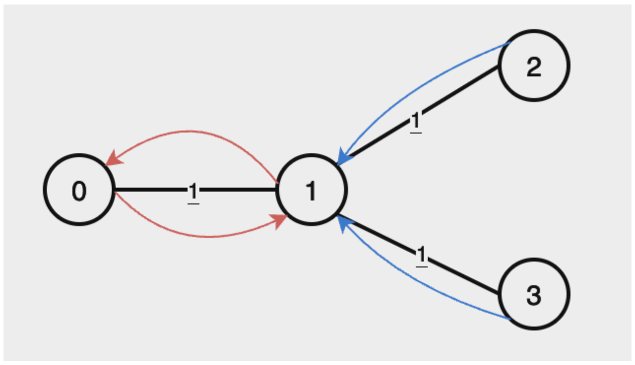

Figure 1: Example: Graph G, $H = \{1, 2, 3\}$, $\alpha = \frac{2}{3}$. Optimal car tour colored in red, friend walking tour in blue, everyone get picked up at node 1. Total cost is $\frac{10}{3}$.

Project Deliverables The project is divided into four parts. In the first part, you have to generate inputs for the problem. In the second part, you have to implement a solver solving the problem. In the third part, you would consider a simpler version and practice dynamic programming. In the last part, you will look at theoretical aspects of the problem.

[1] A path is sequence of nodes where any two consecutive nodes are connected with an edge. A tour is a path that start and ends in the same node and can repeat nodes. A simple tour (or simple cycle) is a tour that does not repeat nodes.

2 Input Generation

Overview In this part, you will have to generate inputs for the problem, that is, create instances of the problem. We would gather everyone's inputs and test your solver in Question 3 on them. You might like to design inputs whose solution is hard to find so only your group can perform well in the test. To do this, you may first think of possible approaches to solve the problem and then design inputs to induce the algorithm to bad solutions.

Input Format The first line of the input file would be a float number α representing the relative cost of driving vs walking.

The second line of the input file contains two integers $|V|$ and $|H|$, separated by spaces, where $|V|$ is the number of nodes in the graph and $|H|$ is the number of homes (friends). The nodes will be indexed from 0 to $|V|-1$.

The third line contains the list of home nodes, separated by spaces.

The following lines have similar structures which specify the adjacency list of the graph.

The first label in a line is the source node index followed by the node degree d. The next d lines are target node indices and integer edge weight. That pattern repeats for all nodes in the graph.

Sample Input: consider the example in Figure 1, the corresponding input file would be

```
0.66667
4 3
1 2 3
0 1
1 1
1 3
0 1
2 1
3 1
2 1
1 1
3 1
1 1
```

Question 2 Generate 4 inputs with different sizes and α in the required format. Your graphs must be connected and satisfy the triangle inequality. Edge weights should be integers. The maximum edge weight is 251219. Float numbers have at most 5 digits. In short, you input should be able to pass the *ck_input* test. You will generate 1 input for each of the following problem categories:

- $\alpha = 0.3$, up to 20 nodes and 10 friends
- $\alpha = 1.0$, up to 20 nodes and 10 friends
- $\alpha = 0.3$, up to 40 nodes and 20 friends
- $\alpha = 1.0$, up to 40 nodes and 20 friends

Name them 20_03.in, 20_10.in, 40_03.in, 40_10.in, respectively.

3 Solve PTP

Overview In this part, you are asked to solve the PTP problem. We know this is demanding, so we give you some hints (actually, possible solutions!) to start with.

Question 3 In *ptp_solver.py* file, implement the function *ptp_solver(G, H, α)* to solve PTP, where inputs are the graph G, home list H, and coefficient α. You should return a list of nodes traversed by the car as well as a list of pick-up locations. More instructions would be given later in this documentation and in the python file.

Your approach must not be the same as in Question 4.2, i.e., you can not solve the problem by using *php_from_tsp*.

You are encouraged to do some research to find similar problems and algorithms to solve them. You can do reductions or/and use the ideas from the solutions. We give you some keywords here: (generalized) traveling salesman (subset tour) problem, shortest paths visiting specified nodes problem, vehicle routing problem...

In case you are struggling to come up with a solution, we provide two possible approaches here:

1. Integer Linear Program (ILP). You can model the problem as an ILP and call a solver to solve it. You can check Miller's paper on 1960 [1] where they formulated TSP as an ILP to get an idea. In Zhang et al (2023)'s paper [2], the authors optimize ride-hailing vehicle routing problem with vehicle-customer coordination where customers can walk to the pick-up locations in an Euclidean space with speed and time constraints. Our problem can also be viewed as a further extension to the generalized TSP problem (see the survey by [3] and the references therein). One can reduce our problem to the generalized TSP and modify the objective function to utilize existing solutions.

2. Greedy algorithm with insert/delete heuristic. In [4], the authors proposed an insert/delete heuristic to solve the traveling salesman subset-tour problem where the salesman has to visit a subset of nodes of the graph with constraints. Here in our problem, we can take the heuristic to build the solution iteratively, each time either to improve the feasibility or reduce the cost.

   Note that once we have a tour T of the car, we can quickly check if it feasible by checking for each friend if any of his neighbors or home is included in the tour. We define the infeasibility measure of tour T as $b(T) = |F| - \sum_{m=0}^{|F|-1} 1(T \cap (\mathcal{N}(m) \cup \{h_m\}) = \emptyset)$, i.e., the number of friends that you can not pick up. Notice that the range of b is $[0, |F|]$. If it is feasible ($b(T) = 0$), the pick-up locations are implicitly defined since we would let friends take the shortest path from their homes (if not already in the tour) to the tour. The cost $c(T)$ of a solution based on T is the sum of α times the total length of the tour and the total walking distance of friends. Therefore, we seek to find a T with minimum $c(T)$ that is feasible. The requirements of T include that it must be a tour starting and ending in node 0 and traverse the neighbors of homes.

   The heuristic algorithm 1 (see pseudocode below) works in following way. We start with an arbitrary tour T and do a local search to improve the current solution. At every step, we change T by either deleting a node from T or adding a new node to T in a way to reduce $c(T)$ the most while maintaining or improving feasibility. This means two possible situations can happen. First, when the current tour is feasible, we select the minimum cost tour that is also feasible. Second, when the current tour is infeasible ($b(T) > 0$), we select the tour with minimum cost among all one-node alternations that have infeasibility less than $b(T)$ (they don't have to be feasible, only need to be better than the current one). Stop when there is no further improvement can be made. Since triangle inequality holds, the total number of changes would be linear in $|V|$ (take it as granted).

Hint: 1. You may want to consider precomputing the all-pair shortest path distances, so you have them ready when executing your algorithm. 2. When calculate $c(T)$, you may relax the constraint that friends must be pick-up nearby and you can let them walk through the shortest path to the tour. But in the end the algorithm would find a feasible tour.

Algorithm 1 PTP Algorithm with Insert/Delete Heuristic

```
$T^1 \leftarrow \{0\}$
$n = |V|$                                        ▷ Number of nodes
for $k = 1, 2, \cdots$ do
    for $i = 1, 2, \cdots, n$ do                  ▷ Compute one node change of $T^k$
        if $i \in T^k$ then
            $T^k_i = T^k.remove(i)$               ▷ Directly remove
        else
            $T^k_i = T^k.least\_cost\_insert(i)$
                ▷ Multiple places to insert. When connecting to a node in the tour use the shortest path to that node. Take the one with minimum $c(T)$. Notice that throughout the process we use the generalized cost function that for infeasible solutions we also take into account a far walking distance instead of putting infinity.
        end if
    end for
    if $b(T^k) = 0$ then
        $i^k = argmin_i \quad c(T^k_i) \quad s.t.\ b(T^k_i) = 0$
    else
        $i^k = argmin_i \quad c(T^k_i) \quad s.t.\ b(T^k_i) < b(T^k)$
    end if
    if $c(T^k) \leq c(T^k_{i^k})$ and $b(T^k) = 0$ then
        Break
    else
        $T^{k+1} = T^k_{i^k}$
    end if
end for
```

4 A Constrained Version

Overview In this part, we consider a simpler version of PTP, namely Pickup from Home Problem (PHP). The problem has the additional constraint that you must pick up your friends at their homes (so we don't need to worry about optimizing over the set of pickup locations).

It is easy to establish NP-hardness of PHP by reducing the Metric Travelling Salesman Problem (M-TSP) to PHP. M-TSP is defined in terms of a graph $G_e = (V_e, E_e)$ that is complete (there is an edge between any two nodes) and triangle inequality holds[2]. It requires to find a tour with minimum total length that is simple (does not repeat nodes). For any instance $G_e = (V_e, E_e)$ of M-TSP, we can construct an instance of PHP with $V = V_e, E = E_e$, where V corresponds to set of the home locations of the party owner and his friends. In this special case of PHP all nodes correspond to home locations and are fully connected with edges.

In the optimal solution of such a PHP instance, each node is visited exactly once: First, we have to visit each node and the solution decides in which order we visit the nodes. Second, no node would be visited more than once since i) the edge weights are nonnegative, ii) the graph is complete, and iii) triangle inequality holds (hence, we can improve the cost if there are loops by taking shortcuts, check it!). Therefore, the solution of PHP is the solution of M-TSP. The transformation is clearly in polynomial time. This completes the reduction. Since M-TSP is known to be NP-hard, so is PHP.

On the other hand, we can reduce PHP to M-TSP. That is, for any instance of PHP, we can transform it to an instance of M-TSP. After getting the solution of TSP from some oracle (which we don't care for now) we can transform it back to the solution of PHP. Then, if we know how to solve TSP, we know how to solve PHP. The transformation can be done in polynomial time by following procedure.

1. Given an instance $(G = (V, E), H)$ of PHP, construct a complete graph $G' = (V', E')$ where $V' = H \cup \{0\}$. For every edge $(u, v) \in E'$, the weight of the edge is the distance of the shortest path from u to v in G.

2. Solve M-TSP on G' to get the tour C'. Note that G' is complete and triangle inequality holds. We introduced a dynamic programming solution to such TSP in lectures.

3. Given C', construct the optimal tour C for PHP by substituting the edges in C' with the corresponding shortest paths in G.

Question 4.1 In *mtsp_dp.py* file, implement the function *mtsp_dp(G)* to solve M-TSP using dynamic programming algorithm introduced in the lectures, where input is a complete graph G with triangle inequality.

If you don't have time to write the DP algorithm, you can call an auxiliary solver to solve TSP but you will get 60% deduction for this problem.

Question 4.2 In *php_from_tsp.py* file, implement the function *php_solver_from_tsp(G, H)* to solve PHP, where inputs are graph G and home list H and output is a list of nodes traversed by the car. You must use the reduction above and solve TSP using **Question 4.1**.

[2]In class we introduced Euclidean TSP where nodes correspond to locations on the map and edge weights correspond to Euclidean distances, hence, triangle inequality holds. Here we consider a more general graph where distances are not necessary Euclidean but triangle inequality still holds

5 Theoretical Questions

Overview In this part, we look into theoretical aspects of the problem including NP-hardness of PTP and approximation ratio of PHP.

Clearly, solving PHP on the same graph gives a feasible solution to PTP. The question is, is it optimal? If not, then how bad can it be?

Question 5.1 Show that PTP is NP-hard.

Hint: Are there values for α for which PHP = PTP (the solution of PHP is obtained by solving PTP)? Since PHP is NP-hard, then PTP is also NP-hard.

In general, we would expect PHP to give a sub-optimal solution for PTP since we don't take the choice of pick-up locations into the optimization. That is to say, in any instance (G, H, α) of the PTP, let C_{php} and C_{ptpopt} be the total cost of the solution obtained from solving PHP (G, H) and the optimal solution of PTP, respectively. Define $\beta = \frac{C_{php}}{C_{ptpopt}}$. Clearly, $\beta \geq 1$. We are interested to know how bad β can become if an adversary is free to choose the parameters of the problem.

Question 5.2 Show that the cost of PHP is at most twice of that of the optimal solution (which we don't know). That is, $\beta = \frac{C_{php}}{C_{ptpopt}} \leq 2$. Also show that this bound is tight, i.e., there is an instance where $\beta = 2$ (at least asymptotically). You can assume $\alpha = 1$ for simplicity.

---

6 Input & Output Format

6.1 Graph Representation

We would use Python package NetworkX to store graphs throughout the project. NetworkX is a very powerful and useful tool to networks studies. It's convenient to modify your graphs such as adding attributes with NetworkX. Install the package here. Check this tutorial to get a quick start. You can find more examples in the handout code of week 7 and week 8.

We will handle input operations and graph constructions for you. We define the API of the functions you should implement which you must obey. The I/O definitions can be found in corresponding question descriptions, section 5.3, and python file comments.

But for your information, and in case you wish to use other representations, which is totally okay if you modify the template correspondingly and submit all your code so we can reproduce your work, we present formats of input files below.

6.2 Input File Format

The first line of the input file would be a float number α representing the relative cost of driving vs walking.

The second line of the input file contains two integers $|V|$ and $|H|$ separated by spaces, where $|V|$ is the number of nodes in the graph and $|H|$ is the number of homes (friends). The nodes will be indexed from 0 to $|V| - 1$.

The third line contains the list of home nodes, separated by spaces.

The following lines have similar structures which specify the adjacency list of the graph.

The first label in a line is the source node index followed by the node degree d. The next d lines are target node indices and integer edge weight. That pattern repeats for all nodes in the graph.

Sample Input: consider the example in Figure 1, the corresponding input file would be

```
0.66667
4 3
1 2 3
0 1
1 1
1 3
0 1
2 1
3 1
2 1
1 1
3 1
1 1
```

6.3 Function APIs

6.3.1 PTP Solver

You are encouraged to come up with different algorithms to solve PTP and compare them. We'd like you to gradually improve your algorithm. PHP solver would be a good start point since you can take pick-up locations into consideration to achieve lower cost as well as use heuristics to gain solutions faster. At last, we would only evaluate your solver in function *ptp_solver* in Question 3. So put your best algorithm there. PTP solvers would have following API.

Input: NetworkX graph G, a list H of home nodes in indices, a float number $\alpha \in (0, 1]$ representing the relative cost of the car per unit of road traveled.

Output: τ, L where τ is the list of indices of the nodes traversed by your car and L is an iterator of (pick-up-locations, people-picked-up) pairs. People would be represented by the index of her home node. Again, your output should be legitimate for following constraints. The indices must be in the graph, i.e., integers from 0 to $|V| - 1$. The tour τ must begin and end at node 0. It can only go through edges that exist in the graph. The pick-up locations must be in τ. Everyone should get picked up nearby their homes.

A sample out of the example in Figure 1 would be

```
    \tau = [0, 1, 0]
    L = {1: (1, 2, 3)}
    return \tau, L
```

6.3.2 M-TSP Solver

You will implement one solver to solve TSP on a metric graph using dynamic programming algorithm in Question 4.1.

Input: NetworkX graph G.

Output: a list of indices of the nodes traversed by the car. The tour must visit each node exactly once. It must begin and end at node 0.

6.3.3 PHP Solver

Essentially, you only need to implement one PHP solver, namely *php_solver_from_tsp* in Question 4.2.

Input: NetworkX graph G and a list H of home nodes in indices.

Output: A list of indices of the nodes traversed by your car. The output must be legitimate. The indices must be in the graph, i.e., integers from 0 to $|V| - 1$. The tour must begin and end at node 0. It can only go through edges that exist in the graph. It must visit every node in H.

6.4 PTP Output File Format

We would store your output for PTP in a file so you can analyze it. The output file corresponding to an input file would have the same name, except with the extension replaced by ".out". For example, the output file for "1.in" would be "1.out".

The first line of the output file would be a list of nodes represent the tour taken by the car, separated by spaces. The nodes would be in the order in which they are visited by the car. The list would start and end at node 0.

The second line would be an integer d represents the number of pick-up locations.

For the following d lines, each line starts with a node index followed by a list of picked up friends, separated by spaces. Your friends are represented by the index of their home nodes.

Sample Output File consider the example in Figure 1, the corresponding output file would be

```
0 1 0
1
1 1 2 3
```

7 Submission & Evaluation

Overview You are encouraged to work in group as working collaboratively is a skill in and of itself. It will also reduce your workload for this demanding project. Only one member of the group needs to submit your solutions to bb. The deadline is 23:59, Dec 19th, 2025.

Evaluation The total point of the project is 100, which is worth 10% of your final grade. You will earn these points as follows.

- 10pts for **Question 2**.
- 20pts for **Question 3**.
- 10pts for **Question 4.1**.
- 10pts for **Question 4.2**.
- 10pts for **Question 5.1**.
- 10pts for **Question 5.2**
- 20pts for proposing a good PTP solver. We will test your PTP solver on all inputs generated by the students. We would calculate the average cost of your solver. You would be scored based on the average cost compared to that of other teams. The score will range from 0 to 20 based on the following:
  - 20pts: your solution performs better than 80% of student submissions.
  - 16pts: your solution performs better than 60-80% of student submissions.
  - 12pts: your solution performs better than 40-60% of student submissions.
  - 8pts: your solution performs better than 20-49% of student submissions.
  - 4pts: your solution performs better than 0-20% of student submissions.
- 10pts for the report.

Submission Details Each group should submit four things: inputs, outputs, code, and report. We will provide you a series of input files. You should run your algorithms in each of them. You need to **submit your output for every input provided** with correct file names. Put the outputs in a separate folder. You also need to **submit your code for solving PHP and PTP**. In part of those, you need to write **a report** containing solutions to the theoretical questions 5.1, 5.2 and approaches you take to solve PTP. For each of the approaches you take, write no more than one page to describe how it works and how it performs. Your report should be in pdf form.

Zip everything into one file and name it with your group ID. A typical submission (say, group_0.zip) would have a minimal structure as in the example (see next page).

Any violation of the requirements will result in a 5-point deduction, applied cumulatively.

```
group_0
├── inputs
│   ├── 20_03.in
│   ├── 20_10.in
│   ├── 40_03.in
│   └── 40_10.in
├── outputs
│   ├── 1.out
│   ├── 2.out
│   └── ...
├── ptp_solver.py
├── mtsp_dp.py
├── php_solver_from_tsp.py
├── report.pdf
└── ...
```

8 Specifications of the Usage of Libraries

You can use any existing packages and solvers. But you have to make sure we can reproduce your work.

9 Academic Honesty

In completing this project, students are expected to adhere to principles of academic honesty. All work submitted must be original and created solely by the individual student or group, unless otherwise specified. Proper citation of sources is required to give credit to the ideas and work of others. Any form of plagiarism, cheating, or dishonesty will result in disciplinary action, which may include a failing grade for the project or course and report to the school.

References

[1] C. E. Miller, A. W. Tucker, and R. A. Zemlin, “Integer programming formulation of traveling salesman problems,” *J. ACM*, vol. 7, no. 4, pp. 326–329, 1960. DOI: 10.1145/321043.321046.

[2] W. Zhang, A. Jacquillat, K. Wang, and S. Wang, “Routing optimization with vehicle–customer coordination,” *Management Science*, vol. 69, no. 11, pp. 6876–6897, 2023. DOI: https://doi.org/10.1287/mnsc.2023.4739.

[3] P. C. Pop, O. Cosma, C. Sabo, and C. P. Sitar, “A comprehensive survey on the generalized traveling salesman problem,” *European Journal of Operational Research*, 2024.

[4] J. Mittenthal and C. E. Noon, “An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint,” *The Journal of the Operational Research Society*, vol. 43, no. 3, pp. 277–283, 1992. DOI: 10.2307/2583718.