Machine Problem 2 (MP2) Documentation

Problem Description

Create an algorithm that will determine the optimal flight pattern (has minimum distance travelled) of a drone given the delivery coordinates of its payload/packages.

MP2 vs Travelling Salesman Problem (TSP)

To describe it simply, the TSP aims to find the shortest possible route in a graph that visits each node once and returns to the starting point. Below is a list of the comparable properties between MP2 and TSP:

1. TSP will return a cycle as a solution whereas MP2 should return a degenerate tree (without considering height). The two paths mentioned should be of minimum distance.
2. The above statement, however, does not mean that the $\text{minPathDistance}_{\text{TSP}}$ is less than $\text{minPathDistance}_{\text{MP2}}$. This is because height is considered in MP2.
3. There is no bound to the number of nodes in TSP while MP2 is limited to at most 50 delivery coordinates.
4. It is assumed (at least in this document) that the given graph is complete and symmetric. This means all nodes are reachable.

Held-Karp Algorithm

Proposed by Held and Karp, the algorithm is a dynamic programming approach to solve the TSP. The main idea of the algorithm is that every subpath (represented as subsets) of a path of minimum distance is itself of minimum distance.

It iterates through every subset S and for every subpath, go through each node N except the starting node and the nodes in S. The minimum distance between the subpath and node is then determined and saved. Lastly, the distances from a set containing all nodes to any other node N is compared to determine the minimum.

The assumptions in the section above is considered in implementing the algorithm. Below are some of the HK algorithm adjustments made to meet MP2's specifications and some of its key properties (HK_{MP2} is the modified version):

1. Set the base case (visiting a node N other than the starting node N_s) to zero. In HK_{TSP}, this is set to the distance between N and N_s due to its cyclic nature.
2. HK_{MP2} represent subsets through bit-masking; the subset of visited nodes are encoded into base-2 system. Bitwise operators are used to remove nodes and compare sets. Due to this, a long int (64-bit integer) is used to allocate space for at most 2^{50} subsets.
3. HK_{MP2} must have a way to determine the endpoints of the path with minimum distance. This is possible because of the assumption that if there exists a path of minimum distance D, then there are two node-subset (set must contain all nodes) pairings that has the same distance D. The two nodes are the endpoints.
4. After determining the minimumPathDistance, HK_{MP2} will add 2H*(numNodes-1) because the vertical distance H from the Cartesian plane is considered (i.e. the drone has to maintain a certain height when travelling).

Complexity Analysis

Space Complexity. The input is an adjacency matrix with space complexity $O(n^2)$. HK algorithm will iterate on every possible occurrence of subset-node pairings which is given by: $(n-1)2^{n-2}$. The space complexity, in total, must be $O(n^2 + (n-1)2^{n-2})$ or simply $O(2^n n)$.

Time Complexity. The program will iterate through every subset-node pairings S-N which has time complexity $O(2^n n)$. For every S-N, the algorithm then iterates through every node not including N_s and the nodes in S which has linear complexity. HK algorithm, therefore, has time complexity of $O(2^n n^2)$.

HK_{MP2} Pseudocode

input: complete and asymmetric graph G with n nodes

output: path with minimum distance

initializations:

minDistance – a 2D (2^n by n) array containing minimum path distance between a subpath and a node; initially contains infinite value INF, base case of visiting one and only one node has zero distance

S – the set containing all n nodes

methods:

$eDist_{a,b}$ – edge distance from node a to node b

$pDist_{a,b}$ – minimum distance from subpath a to node b (i.e. minDistance[a][b])

algorithm HK_{MP2}(G):

```
    initialize: minDistance
    for every subset C of S excluding null sets:
        for every node N in G:
            ignore if N is in C or C has one element
            for every node M in G:
                initialize: subset D = C \ {M}
                minDistance[C][M] = min(pDist_{C,M}, pDist_{D,N} + eDist_{N,M})
    initialize: source = 0, destination = 0, distance = INF
    for every node N in G:
        distance = min (distance, minDistance[S][N])
        ignore if distance is not changed:
        source = N
    for every node N in G:
        if distance == minDistance[S][N]:
            ignore if source == N
            destination == N
    return source, destination, distance
```