

A brute-force solution to problem “t_{sp}”

As noted in the “Five example optimization problems” handout, problem t_{sp} is intrinsically a discrete optimization problem. In fact it is a *combinatorial* and graph-theoretic optimization problem. Such problems are a topic in MATH 663 Graph Theory. For that reason, and also because t_{sp} fits neither with the continuous optimization problems of the rest of the course, nor the techniques in the textbook, we will not cover it further in MATH 661 Optimization, beyond the brute-force solution t_{sp}.m on this document. *There is no claim that this code represents a good approach*, but it does solve the problem. Please see a course or book on combinatorial optimization or graph theory for more information on this kind of problem.

The MATLAB/OCTAVE code t_{sp}.m starts with a generally-useful and basic idea, namely that one may represent an edge-weighted graph¹ by a *matrix*. First, the vertices (cities) A, F, J, N, S, W have indices 1, 2, 3, 4, 5, 6, respectively, in this data structure. Then the matrix $M = \text{edgew}$ has entry $m_{ij} = -1$ if there is no flight between city (vertex) i and city j ; otherwise m_{ij} is the cost of the flight from city i to city j .

If the seven-city itinerary $p = i_1 i_2 i_3 i_4 i_5 i_6 i_7$ is feasible, in the sense that the corresponding flights exist, then the cost of p is the sum of the six flights,

$$C(p) = m_{i_1 i_2} + m_{i_2 i_3} + m_{i_3 i_4} + m_{i_4 i_5} + m_{i_5 i_6} + m_{i_6 i_7} = \sum_{k=1}^6 m_{i_k i_{k+1}}.$$

This cost is computed by a `for` loop:

```
C = 0; % C = cost of path (or -1)
for k = 1:6
    w = edgew(p(j,k),p(j,k+1));
    if w < 0
        C = -1;
        break
    else
        C = C + w;
    end
end
```

If the itinerary is not feasible we “break” out of the `for` loop and move to the next case.

The above explains how to find the optimal itinerary among the feasible ones. How to generate all feasible itineraries? The brute force approach here is to pre-generates all itineraries using MATLAB/OCTAVE’s `perms` function, which generates all permutations of a given vector. Because the itineraries start and end in city S , we generate the permutations of the other cities (`[1 2 3 4 6]`) and then prepend and append $S = 5$. Then, in a big `for` loop, we test whether the itinerary is feasible and compute its cost if so. Note the small

¹A *graph* is a collection of vertices with *edges* which can be defined as pairs of vertices. It is one of the basic objects of (discrete) mathematics.

anonymous function `printcase`, taking three arguments, whose only purpose is to print out an itinerary. Running the code finds only four feasible itineraries and identifies one of the two optimal solutions:

```
>> tsp
feasible   SANFWJS : 1250
feasible   SFNAWJS : 1200
feasible   SJWANFS : 1200
feasible   SJWFNAS : 1250
optimal    SFNAWJS : 1200
```

`tsp.m`

```
% TSP Setup and solve a traveling salesperson problem

label = ['A' 'F' 'J' 'N' 'S' 'W'];
edgew = [ -1 100 100 150 250 150; % A
          100 -1 150 200 300 250; % F
          100 150 -1 -1 200 200; % J
          150 200 -1 -1 -1 -1; % N
          250 300 200 -1 -1 -1; % S
          150 250 200 -1 -1 -1]; % W

% generate permutations
notS = [1 2 3 4 6]; % not including S
N = factorial(5); % this many permutations of notS
p = [5*ones(N,1) perms(notS) 5*ones(N,1)]; % start and end in S

% print one case on a single line
printcase = @(a, p, s) fprintf('%s %c%c%c%c%c%c : %d\n', ...
    a, label(p(1)),label(p(2)),label(p(3)), ...
    label(p(4)),label(p(5)),label(p(6)),label(p(7)), s);

% evaluate cost of each (feasible) permutation
mincost = 1.0e100; % bigger than any
for j = 1:N
    C = 0; % C = cost of path (or -1)
    for k = 1:6
        w = edgew(p(j,k),p(j,k+1));
        if w < 0
            C = -1;
            break
        else
            C = C + w;
        end
    end
    if (C > 0) % report if feasible; update optimal
        printcase('feasible ',p(j,:),C)
        if C < mincost
            mincost = C;
            minp = p(j,:);
        end
    end
end
end
printcase('optimal ',minp,mincost)
```