In problem 4.2, you questioned how my algorithm can avoid exponential search. I will introduce it here in details.

The original problem is a decision problem to determine if a given graph is 2-stable.

One significant observation is that in the residual graph, if there is no path from sink to source node with number of equal or less than 2 saturated edges, capacity increasing on any two edges won’t generate a new augmenting path from source to sink node. Therefore, the graph is 2-stable.

Based on above observation, the decision problem can be transformed to optimization one, which aims to find the minimum number of saturated edges in all paths from sink to source node in the residual graph.

The general idea of my DFS variant algorithm is based on memorial search, which is kind of dynamic programming, but trade space consumption for time efficiency.

For every vertex in the graph, I define a state variable, for example, f[i] to denote the minimum number of saturated edges of searching paths starting from i vertex in the searching tree. (this part is mentioned in my solution)

Iterative relation of f[i] is like:

f[i]= min { f[j]+ w[i,j] }, j is i's child vertex in searching tree. If edge (i,j) is saturated, w[i,j]=1. Otherwise, w[i,j]=0;

Initially, all values of f[i] are set as null. When deep-searching vertex i, if f[i] is not null, it means that the sub-tree rooted at vertex i has been traversed before. Based on f[i], we can update the current vertex's f[] value without need to search the following paths again. If f[i] is null, it still needs to search the following in-visited paths. This is the key point to avoid repeated path searching and keep the DFS running in O(|V|+|E|).

Because we only need to check if the graph is 2-stable or not. After the DFS, we just scan the f[] to see if some of them is smaller than 3. If there is such f[i], the graph is not 2-stable. Otherwise, the graph is 2-stable. This step only cost O(|V|) time.