Report: this report includes:

- Dynamic programming algorithms in the form of the recurrences.

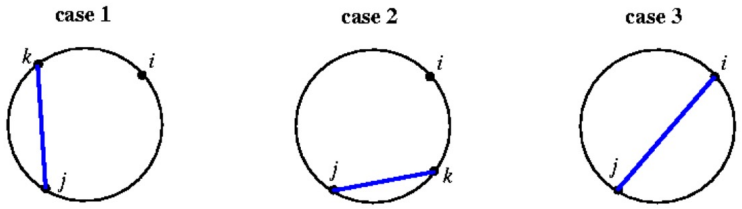

Figure 1: Different cases

Let dp[i][j], $i \leq j$, denote the number of chords in the maximum planar subset

  - MPS
    Base case: $i = j$
    dp[i][j] = 0
    $i < j$:
    case1:$k < i$ or $k > j$
    dp[i][j] = dp[i][$j-1$]
    case2:$k > i$ and $k < j$
    dp[i][j] = max{dp[i][$k-1$]+1, dp[i][$j-1$]}, $k = j-1$
    dp[i][j] = max{dp[i][$k-1$]+dp[$k+1$][$j-1$]+1, dp[i][$j-1$]}, otherwise
    case3:$k = i$
    dp[i][j]=1, $i = j-1$
    dp[i][j]=dp[$i+1$][$j-1$], otherwise
    case4: there's no chord starting from j
    dp[i][j] = dp[i][$j-1$]
  - LMPS
    Base case: $i = j$
    dp[i][j] = 0
    $i < j$:
    case1: there's no other node having the same label of j.
    dp[i][j] = dp[i][$j-1$]
    case2: there are n nodes have the same label of j. We denote them as $\{k_1, k_2, ...k_n\}$
    This case can split into n cases. Each case we delete other $n-1$ chords and only consider k_i, $i = 1, 2, ...n$. And this case can further split into three cases in Figure 1. We can use the same form of the recurrences in MPS to get the maximum planar subset. Then we take the maximum of n cases.

- Running time and space complexity analysis of your algorithms.

  Input size: num_nodes=n

  MPS Time Complexity Analysis:

  Maximum_planar_subset function:the double loop iterates over all pairs of i and j, leading to a time complexity of O(n^2). The recursive function print_chords is called to reconstruct the chord edges contributing to the maximum planar subset.The recursion depth is at most num_nodes, and within each recursion, there are constant time operations.

Table 1: **mps and lmps running time**

number of nodes	mps running time	lmps running time
8	0.00050306	0.00055480
9	0.00046802	0.00041294
10	0.00096797	0.00047398
12	0.0014041	0.0013762
15	0.00092220	
16	0.00073600	0.0014288
32	0.0015009	0.0033138
50	0.0017741	0.0065460
64	0.0019210	0.010856
128	0.0039401	0.057390
256	0.0098612	0.31480
512	0.038240	2.4455
1024	0.14432	19.866
2048	0.59789	165.88
2500		312.11
4096	2.5872	
8192	11.650	
20000	70.069	
25000	111.80	

Total time complexity is O(n^2).

MPS Space Complexity Analysis:

A 2D array of size n^2 is used to store intermediate results during dynamic programming. The size of chord_table is n. The size of array result is no larger than n^2.

Total space complexity is O(n^2).

LMPS Time Complexity Analysis:

Maximum_planar_subset function:k iterates all nodes with same labels of j, and the double loop iterates over all pairs of i and j, leading to a time complexity of O(n^3). The time complexity of recursive function print_chords is no larger than n^3.

Total time complexity is O(n^3).

LMPS Space Complexity Analysis:

A 2D array of size n^2 is used to store intermediate results during dynamic programming. The size of chord_table is n. The size of array temp is no larger than n. The size of array result is no larger than n^2.

Total space complexity is O(n^2).

- Plot actually measured running time (vs the number of nodes) on the input instances for both cases(Figure 2 and Figure 3).
- Discuss how empirical running time reflects the running time from theoretical analysis.

  For mps, the algorithm is expected to have time complexity O(n^2), observing a roughly quadratic increase in running time as the input size grows supports the theoretical analysis. For lmps, the algorithm is expected to have time complexity O(n^3), observing a roughly cubical increase in running time as the input size grows supports the theoretical analysis. Empirical running time may have some

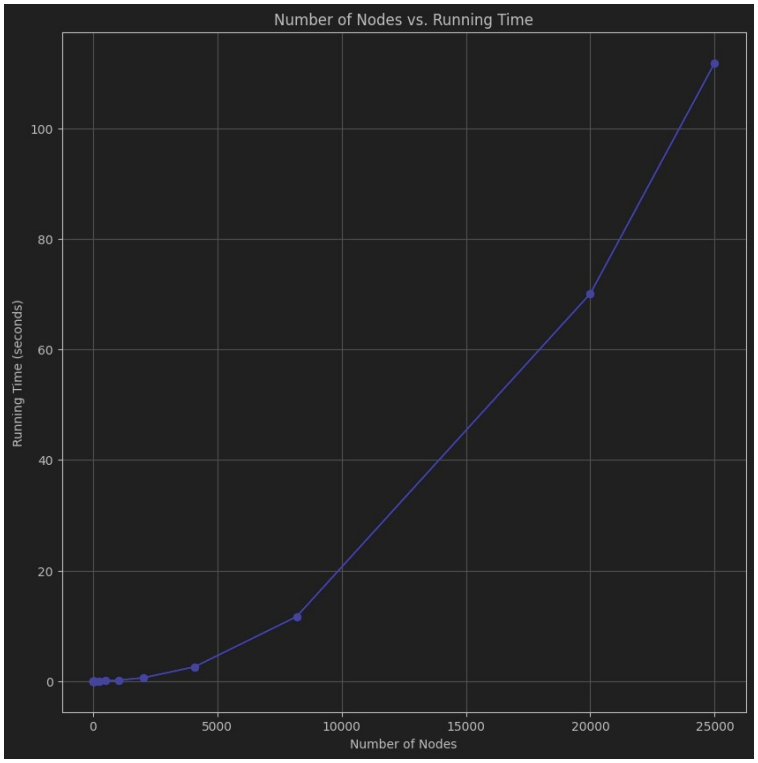

Figure 2: mps measured running time

small deviations because of the impact of compiler optimizations, hardware architecture, and other factors that may influence the actual running time.

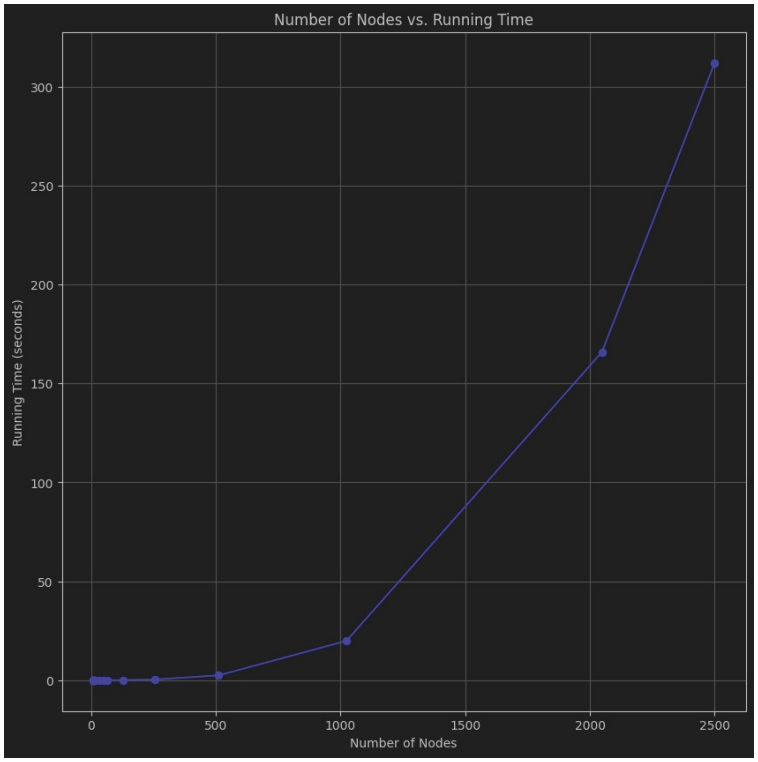

Figure 3: lmps measured running time