Given a graph,find the edge-disjoint and vertex disjoint subgraphs where subgraphs are complete graphs.

Shiv Pratap Singh, Nazish Tabassum, Arqum Ahmad

***Abstract*—Edge-disjoint: Two graphs not having any edge common. Mathematically, if G1 and G2 are two graphs, then G1∩G2 = ∅ and G1⊕G2=G1∪G2**

Vertex-disjoint-Two graphs not having any vertex common. Mathematically,if G1 and G2 are two graphs , then G1∩G2 is empty.

***Index Terms*—**

I. Motivation

Vertex disjoint and edge disjoint algorithms are mainly used in chemistry to find chemicals that shows some similar property than others or chemical that match with target chemicals.

II. Algorithm

A. All maximal cliques

Input: A graph in the form of adjacency matrix

Output: A set C containing edge disjoint maximal cliques in graph

EdgeDisjointClique(R, P, X):

```
1  if P and X are both empty:
2      report R as a maximal clique
3  for each vertex v in P:
4      EdgeDisjointClique(R ∪{v}, P ∩ N(v), X ∩ N(v))
5      P := P \ {v}
6      X := X ⋃{v}
```

B. Edge disjoint maximal cliques

Input: A graph in the form of adjacency matrix

Output: A set C containing edge disjoint maximal cliques in graph

EdgeDisjointCliques

```
1  Find isolated vertex in graph
2      pushAllIsolatedVertexIn(C)
3  PrepareEdgeSetFromGraph
4  while(NotEdgeSetEmpty)
5      (u, v) = EdgeSet.pop()
6      cliqueSet.insert(u, v)
7      ExpandCliqueSet(v, cliqueSet)
8      cliqueSet.clear()
```

ExpandCliqueSet(v, cliqueSet)

```
1  For j iterate over N(v)/cliqueSet
2      If JIsExpandable(j, cliqueSet)
3          CliqueSet.insert(j)
4  CliqueSetRemoveAssociatedEdges(cliqueset)
5  C.insert(cliqueset)
```

JIsExpandable(j, cliqueSet)

```
1  For i iterateOverCliqueSet
2      IsNeighbour(i,j)
```

CliqueSetRemoveAssociatedEdges(cliqueset)

```
1  RemoveAssociatedEdgesfromGraph
2  RemoveAssociatedEdgesfromEdgeSet
```

C. Vertex disjoint maximal cliques

Input: A graph in the form of adjacency matrix

Output: A set C containing vertex disjoint maximal cliques in graph

VetexDisjointCliques

```
1  Find isolated vertex in graph
2      pushAllIsolatedVertexIn(C)
3  PrepareEdgeSetFromGraph
4  while(NotEdgeSetEmpty)
5      (u, v) = EdgeSet.pop()
6      cliqueSet.insert(u, v)
7      ExpandCliqueSet(v, cliqueSet)
8      cliqueSet.clear()
```

ExpandCliqueSet(v, cliqueSet)

```
1  For j iterate over N(v)/cliqueSet
2      If JIsExpandable(j, cliqueSet)
3          CliqueSet.insert(j)
4  CliqueSetRemoveAssociatedVertex(cliqueset)
5  C.insert(cliqueset)
```

JIsExpandable(j, cliqueSet)

```
1  For i iterateOverCliqueSet
2      IsNeighbour(i,j)
```

CliqueSetRemoveAssociatedEdges(cliqueset)

```
1  RemoveAllAssociatedEdgesfromGraphForEachVertexInCliquesSet
2  RemoveAllAssociatedEdgesfromEdgeSetForEachVertexInCliquesSet
```

III. Description

A. AlgorithmA

We proposed two algorithms to find the complete subgraphs of a given graph. In this algorithm , which maintains a set of maximal cliques. To find the maximal clique recursive backtracking algo is implemented.We maintain three sets R,P,X.

R : possibly a maximal clique.

P : holds the vertices adjacent to every vertex currently in R,none/few/all when added to R makes it maximal.

X : contains nodes already in some clique or processed(To remove duplicates.

Initially P contains all the vertices of graph. choose vertex vfrom P and push it in R. Choosing of vertex v such that the degree of v is highest among all the vertices in P. Update P = P ∩ N(v) where N(v) = all neighbours of v. Recursively Call function maximalCliques(R,P,X) until P ∪ X is an empty set, else backtrack for other vertex. If a vertex has already been part of some other clique,pushed it in X to avoid duplicates. If all the vertex has been visited, return set R having vertices of maximal clique. And finally, from set R cliques not having any common vertex pushed to vertex-disjoint Set and cliques not having any edges common pushed to edge-disjoint set.

B. AlgorithmB

In this we make edgeSet from adjacency matrix then we simply add consider this edge to be one clique and we expand this clique using ExpandCliqueSet function which add neighbours of any end point of edge(u,v) If and only if this neighbour is adjacent to each element in cliqueset. At last we remove edges in cliqueSet from graph and add this cliqueset to be our clique set C.

C. AlgorithmC

In this algorithm we remove the vertex in cliqueSet instead of removing edges to find vertex disjoint cliques.

IV. Implementation and Results

The algorithm has been implemented in c++ language. Test case are generated manually using c++ library. For particular input size running time of input is calculated. The tool used for plotting graph is gnuplot.

V. Plot of time taken vs input size

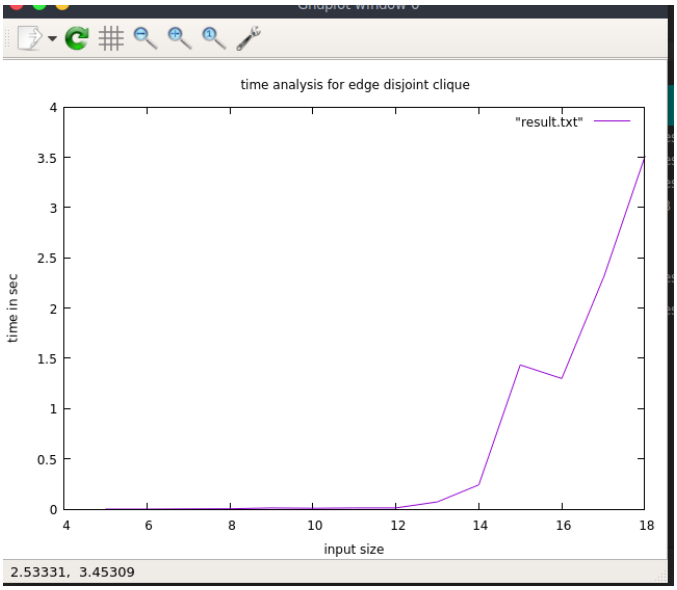

Fig. 1. Time analysis of All maximal cliques algorithm

VI. Conclusion

In this problem,we assume that the adjacency matrix of a given graph has been provided. Final outputs are sets of subgraphs which are vertex-disjoint and edge-disjoint with condition that subgraphs are complete. To achieve the final sets of output,we in between maintain an another set which keep track of complete subgraphs in first algo and track of maximal complete subgraphs in second algo. Now if cliques in these set are vertex disjoint, push it in vertex disjoint set or if edge-disjoint,push it in edge disjoint set and finally return these sets. The time complexity for finding the cliques in ALGORITHM1 is O(max((E*V),(V*V)) The time complexity for finding the maximal cliques in ALGORITHM2 is O(3n/3) where n= Total number of vertices.

VII. Refrences

1. https://en.wikipedia.org/wiki/Clique$_problem$
2.$Subgraphs, Paths, and Connected Graphs - Springer$
3.$https://en.wikipedia.org/wiki/Bron$