Extraction of largest complete subgraph for a given graph

Maharshi Roy, Rajat Kumar Sahu, Amrit Daimary

ism2014006@iiita.ac.in
ihm2014501@iiita.ac.in
irm2014001@iiita.ac.in

Abstract—*This paper provides an insight into the various procedures to extract the largest complete subgraph if present in a given graph. Particularly, the* ***Bron-Kerbosch*** *algorithm and* ***Branch & Bound*** *methods are discussed in detail.*

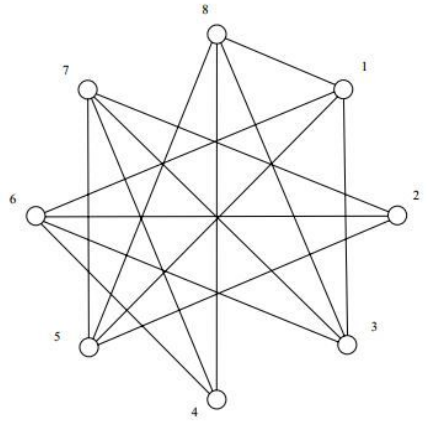
Keywords — *Maximal Clique, Largest Complete subgraph.*

MOTIVATION

Largest Complete subgraphs or Maximal Cliques as they are known otherwise and their extraction or mining from an otherwise given graph is an interesting problem that finds its applications in numerous real life problems. The task, however being **NP-Hard,** does not forbid it from being repeatedly used as a subroutine in solving even more tedious real life problems. e.g. (1) Clique detection has traditionally been a very important tool in chemoinformatics, where one problem is to identify common substructures between a collection of molecules known to possess certain pharmacological properties. (2) Finding the maximal clique also gives us a lower bound. on the chromatic number of the graph. i.e, at least that much distinct colors are definitely required in a graph coloring problem.

METHODOLOGY AND APPROACH

We shall discuss 2 methods for finding maximal cliques in this paper. First one goes by the name of **Bron-Kerbosch** Algorithm, and the second one is a **Branch & Bound** method.



Bron-Kerbosch Algorithm

The basic form of the Bron–Kerbosch algorithm is a recursive backtracking algorithm that searches for all maximal cliques in a given graph G. More generally, given three sets R, P, and X, it finds the maximal cliques that include all of the vertices in R, some of the vertices in P, and none of the vertices in X. In each call to the algorithm, P and X are disjoint sets whose union consists of those vertices that form cliques when added to R. In other words, P ∪ X is the set of vertices which are joined to every element of R. When P and X are both empty there are no further elements that can be added to R, so R is a maximal clique and the algorithm outputs R.

The recursion is initiated by setting R and X to be the empty set and P to be the vertex set of the graph. Within each recursive call, the algorithm considers the vertices in P in turn; if there are no such vertices, it either reports R as a maximal clique (if X is empty), or backtracks. For each vertex v chosen from P, it makes a recursive call in which v is added to R and in which P and X are restricted to the neighbor set N(v) of v, which finds and reports all clique extensions of R that contain v. Then, it moves v from P to X to exclude it from consideration in future cliques and continues with the next vertex in P.

Algorithm:-

.

```
BronKerbosch(R, P, X) {
    if P and X are both empty
        report R as maximal clique
    for each vertex v in P {
        BronKerbosch(R∪{v}, P∩N(v), X∩N(v))
        P = P \ {v}
        X = X ∪ {v}
```

The worst case time complexity of this method has been found to be of the order $O(3^{n/3})$.

Branch & Bound Method

A branch & bound recursive algorithm for finding maximal cliques works as follows. For each recursive, we have an initial vertex to start from. For each call, we keep a separate set of remaining vertices, and in each call stack, we attempt

to add a vertex to the set of included vertices if it satisfies the full connectivity condition, thus maintaining the complete connectivity property. If we are unable to find any such vertex, we compare the size of the clique found to the best solution and update it if the current size exceeds it.
The time complexity of the algorithm has been found out to be ***O(2ⁿ * n)***.

Algorithm:-

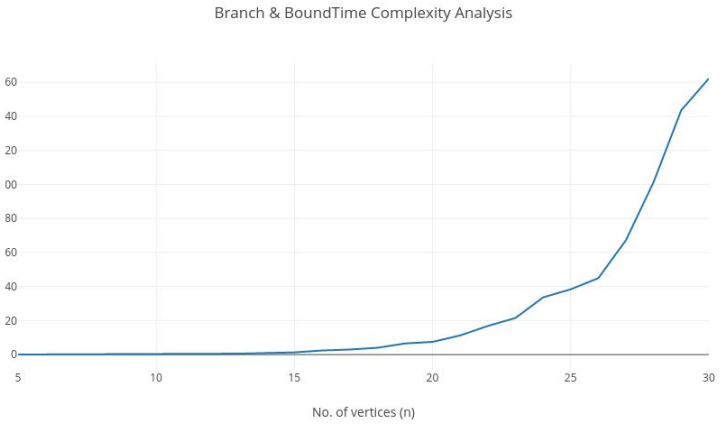
```
bool check(U,v) {
    bool ans=true
    for all vertex w in U {
        ans = ans & (G[w][v])
    }
    return ans
}

best = 0
soln = ϕ
find_max_clique(U,trace) {
    flag=false
    for all vertex v from [trace+1,n) {
        if (check(U,v) == true) {
            flag=true
            U = U ∪ {v}
             find_max_clique(U,v)
            U = U \ {v}
        }
    }
    if (flag==false) {
        if (|U| > best) {
            best = |U|
            soln=U
        }
    }
}

main() {
    for all vertex v in G {
        U = U ∪ {v}
        find_max_clique(U,v)
        U = U \ {v}
    }
    print <best, soln>
}
```

IMPLEMENTATION & RESULTS

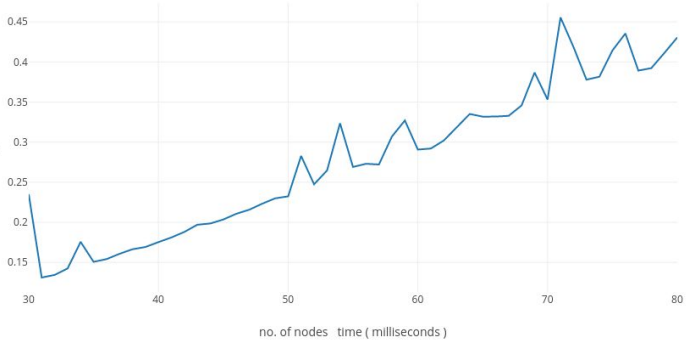
The algorithm was implemented in C++ and Time complexity analysis were performed on all the three methods, and line-graphs were obtained for varying inputs. All these operations were done in GNUPlot. All the tasks was performed in a 64-bit machine (x86-64 architecture). The line-graphs for the various methods are given as follows:-

Time Complexity Plot:-

Branch & Bound Time Complexity Analysis



Bron-Kerbosch Time Complexity Analysis



REFERENCES

[1] Bron-Kerbosch Article from Wikipedia

[2] Patric R.J. Ostergard, *A fast algorithm for the maximum clique problem*, Discrete Applied Mathematics 120 (2002) 197–207

[3] Etsuji Tomitaa, Akira Tanakaa, Haruhisa Takahashi, *The worst-case time complexity for generating all maximal cliques and computational experiments*, Theoretical Computer Science 363 (2006) 28 – 42