Organic Defective Clique Generation Algorithm for solving Maximum s-defective Clique Problem

Sohom Chatterjee, UIN: 729007535
Soham Das, UIN: 130004757

May 2020

1 Introduction

The study of complete subgraphs has been an integral part of mathematics even before the "clique" terminology had been introduced. A clique in a graph is a set of pairwise adjacent vertices, that is, the graph induced by a clique is complete with all possible edges [1]. The term "clique" and the problem of formulating clique-detection algorithms both dawn from the social sciences, where complete subgraphs are used to model social cliques, groups of people who all know each other. The term first arose in [2] where graphs were used to model social networks, and adapted the social science terminology to graph theory. This problem has tremendous applications in chemistry, bioinformatics, etc.

However, the constraint enforcing the existence of all possible edges between a group of vertices seems to be overly restrictive and impractical, mainly because real-life subgroups may not be perfect and could be missing a few edges. The need for relaxations of the clique model also arises in practice when dealing with massive data sets which are error prone, resulting in false or missing edges. Many clique relaxation models have been introduced in the literature over time, namely, the s-clique model [3], the sociometric clique of diameter s [4], the s-club [5], the s plex [6], etc. A thorough analysis of different clique relaxation concepts and their mathematical programming formulations have been provided in [7].

In this work, we will focus on the s-defective clique model that was introduced in the context of analysis of protein interaction networks [8]. The defective clique problem can be formally stated as follows,

The maximum s-defective clique problem: Given a simple undirected graph $G = (V, E)$, an *s-defective clique* is a maximum-cardinality subset of vertices that induces a subgraph that has all but at most s edges.

2 IP Formulation

The integer programming formulation for the s-defective clique problem can be written as:

Let $\mid V \mid = n$. Let x_i be a binary variable, which assumes value 1 if vertex i is a part of the defective clique S, and 0 otherwise. Our decision variables are these x_i, which decide which vertices are going to be a part of the defective clique, where $i \in V$. Let z_{ij} be 1 if a pair of vertices i and j are both part of S, and 0 otherwise. Let s be the maximum number of defects we can allow in this clique. Hence, the optimization problem is,

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & \sum_{(i,j)\notin E} z_{ij} \leq s \\ & z_{ij} \geq x_i + x_j - 1,\ z_{ij} \geq 0\ \forall i, j \in V \\ & x_i \in \{0, 1\}\ \forall i \in V \end{aligned}$$

If $(i, j) \notin E$, and $z_{ij} = 1$, we are including a pair of vertices that are not connected by an edge in our desired subgraph S, which is a defect f or the clique. Constraint 1 ensures that we can only tolerate at most s such defects.

The idea of z_{ij} being 1, only if both x_i and x_j are included in the clique can be mathematically formu-

lated as $z_{ij} = x_i x_j$, which however is a quadratic in our decision variables. In order to linearize this, we use the constraints 2 and 3. If both i and j are a part of the subgraph S, then the value of x_i and x_j are both 1, and constraint 2 reduces to $z_{ij} \geq 1$, which makes constraint 3 redundant. Since we are maximizing the number of vertices to be included in the subgraph S, while also having an upper bound for the sum of z_{ij}s, $\forall\ (i,j) \notin E$ in constraint 2, this automatically restricts the corresponding z_{ij}s to not exceed 1.

Thus we can see that the above IP formulation correctly represents the problem in question.

3 Theoretical Analysis

3.1 Proof for NP Hardness

The theorem proposed by Yannakakis in [9], is a standard way of proving NP hardness of several graph properties. It states that,

Theorem 3.1. *The problem of finding a maximum cardinality subset of vertices that satisfies a given property π, which is non trivial, interesting and hereditary on the induced subgraph is NP Hard.*

Here, we need to define some properties.

Definition 1. *We say that a property π of a subset of vertices is hereditary in induced subgraphs if C satisfies π implies that for any C' $\subseteq$ C, C' also satisfies π.*

Definition 2. *A graph property π is non-trivial if it is satisfied by $C = \{r\} \subseteq V$, but not satisfied by every subset of vertices for every graph.*

Definition 3. *A graph property π is interesting if there are arbitrarily large graphs having this property.*

For the s-defective clique problem, the property of interest π can be s-defectiveness, and hence we can restate the problem as,

The maximum s-defective clique problem: Find the maximum cardinality of subset of vertices that satisfies the property of s-defectiveness, ie. it is a clique with at most s-defects.

We can easily see that any induced subgraph of an s-defective clique will also be an s-defective clique, hence it is hereditary on induced subgraphs. This property is also non-trivial, as we can construct graphs which do not satisfy this property for a certain s. This property is also found in arbitrarily large graphs. Hence this property conforms to the definition of π.

Hence this is an NP-Hard Problem.

3.2 Polynomial Time Solvable Cases

There are certain network topologies under which the maximum s-defective problem is polynomial time solvable. Shirokikh in his dissertation, [10] mentions that,

Theorem 3.2. *For a fixed nonnegative integer s, the maximum s-defective clique problem is polynomial-time solvable on planar graphs.*

3.3 Bounds on Optimal Solution

Let the clique number of the graph $G = (V, E)$, that is the cardinality of the maximum clique of the graph be represented as ω_G. Since any s-defective clique is a clique relaxation, a lower bound on the maximum s-defective clique would be ω_G.

In the best case, an s-defective clique can be grown from a max-clique by adding s nodes such that each contribute only one defect. Hence, an upper bound on the maximum s-defective clique can be given as $\omega_G + s$, where s is the number of allowable defects.

The heuristic processes described in the next section also provide tighter lower bounds than the clique number of the graph.

4 Algorithmic Solutions

In this project we propose a few heuristic solutions for generating feasible s-defective cliques as well as an exact approach to generate the maximum s-defective clique.

The basic approach used in these algorithms is to add a certain number of edges using some heuristic and then solving the maximum-clique problem on this new graph to find an s-defective clique for the original graph.

4.1 Random Edge Addition

Here we add s edges to our graph randomly. We use an Integer Programming formulation for the maximum clique problem in Gurobi for the resulting graph. This heuristic generates feasible s-defective cliques and provides a valid lower bound. It is also computationally comparable to merely solving the maximum clique problem.

4.2 Degree-Rank Edge Addition

Since, intuitively we feel that an almost clique would have a lot of high degree nodes where some are not connected to each other, it would make sense to rank the nodes of the graph in terms of their degree and keep adding the allowable number of defects (edges) in this hierarchy.

This leads us to our second Heuristic, where we sort the nodes and then we find pairs of highest order nodes, that are not connected in the original graph and add edges between them in this order.

4.3 Connection-Potential Edge Addition

We define the term *Connection Potential* as,

Definition 4. *Given a maximal clique C in a graph G=(V,E), and a node i which does not belong to that clique, the connection potential, C_p of that node i to the maximal clique C is given by,*

$$C_p(i, C) = \frac{1}{\sum_{j \in C} x_{ij}},$$

where $x_{ij} = 1, \forall (i, j) \notin E$

Essentially this represents the affinity of the node i to be included in the maximal clique. The connection potential of a node which only lacks one edge to be included in the maximal clique will have $Cp = 1$. If more edges are required, the denominator will go up and the connection potential will decrease. Thus ideally, $C_p \in (0, 1]$

We will use this connection potential to identify the best nodes to add to the maximum clique in the graph and thereby grow it into an s-defective clique.

This idea can also be extended to provide an exact approach to finding the maximum s-defective clique by adding nodes to all maximal cliques in the graph, identified using the classical CLIQUES algorithm, discussed in [11].

4.4 Organic Defective Clique Generation Algorithm

This algorithm is inspired by the CLIQUES algorithm to find all maximum cliques as introduced in [12]. Consider a graph $G = (V, E)$. First we introduce a global variable Q that comprises of a set of vertices that constitute a complete subgraph. We have a global variable $deft$ that will store the list of all the defective cliques we find paired with their unrealized defect. We also have a global variable CQ contain the list of all maximal cliques we find using this algorithm.
The algorithm begins by letting Q be an empty set and expands Q step-by-step using a recursive procedure DFS to V and its succeeding induced subgraphs to search for complete subgraphs of larger cardinality, till a maximal subgraph is obtained. At this point the value of Q is appended to the list of cliques, CQ.
Once a clique is formed, the global variable $deft$ will store an imprint of the clique and the number of unrealized defects which is s at this point. Function calls return and we move to the earlier stage of the recursion to find a suitable element that we can add to the maximal clique already generated. This procedure is described in Algorithm 1. We keep adding elements to the clique till its number of unrealized defects become 0 or we run out of suitable elements.

5 Computational Results

We have implemented the algorithms in Python 3.7, on a system of 8GB RAM, Intel(R) Core(TM) i5-4210U CPU @1.70GHz. We have run our simulations on ER graphs generated with connection probability, p= 0.4. For a chosen cardinailty of the ER graph and a chosen number of allowable defects, we run all the algorithms at our disposal along with the standard branch and bound maximum clique algorithm using Gurobi 9.0. All our 2-stage heuristic algorithms depend on this integer programming formulation for the maximum clique problem solved in Gurobi for the second stage implementation.

Algorithm 1 ODCLIQUES(G)

```
procedure ODCLIQUES(G)
    Q ← {}                                          ▷ constitutes a clique
    deft ← []                                       ▷ set of all defective cliques
    s ← n                                           ▷ number of allowable defects
    CQ ← []                                         ▷ set of all cliques
    \* V are the nodes of G\*
    DFS(G,V,V)
procedure DFS(G, SUBG, CAND)
    if SUBG == φ then
        Append (Q, s) to deft
    else
        u = a vertex in SUBG that maximizes |CAND ∩ N(u)|
        n ← N(u)
        for ele in CAND-n do
            Q ← Q ∪ {ele}
            Q_copy ← Q
            SUBG_q ← SUBG ∩ N(ele)
            CAND_q ← CAND ∩ N(ele)
            DFS(G,SUBG_q,CAND_q)
            CAND ← CAND − {ele}
            for i in N(ele) do
                key = the defective clique set of max cardinality in deft of which Q_copy is a subset
                defects = the number of remaining defects in the key
                \* deft is a list that contains the defective cliques and their unrealized defects\*
                for ele2 in N(ele)-key do
                    req = # of defects realized by adding ele2 to clique key
                    if req ≤ defects then
                        Q_copy ← Q_copy ∪ {ele}
                        defects = defects - req
                        Append the tuple (Q_copy, defects) to deft
                        key ← key ∪ {ele}
            if len(Q) ≥ 1 then
                Append Q to CQ
            Q ← Q − {ele}
```

Graph	Gurobi IP	OD Cliques	Degree-Rank	Random	C_p	Max Clique Gurobi
$\|V\|$ =20, s=3	6.2	6.0	5.1	4.8	5.6	4.6
$\|V\|$ =30, s=5	7.7	7.5	6.0	5.5	6.9	5.4
$\|V\|$ =40, s=7	9.1	8.4	6.5	5.9	8.0	5.9
$\|V\|$ =50, s=9	10.0	9.2	6.9	6.2	8.5	6.1
$\|V\|$ =100, s=5	10.0	10.0	8.0	8.0	9.0	8.0

Table 1: Average Max Cardinality of Defective Cliques

Graph	Gurobi IP	OD Cliques	Degree-Rank	Random	C_p	Max Clique Gurobi
$\|V\|$ =20, s=3	0.1062	0.0855	0.0078	0.0047	0.0109	0.0031
$\|V\|$ =30, s=5	0.7703	0.6500	0.0141	0.0078	0.0203	0.0141
$\|V\|$ =40, s=7	5.0078	3.3559	0.0266	0.0266	0.0640	0.0422
$\|V\|$ =50, s=9	25.5218	12.6675	0.0812	0.1187	0.1422	0.1109
$\|V\|$ =100, s=5	412.3594	1186.4055	1.1250	1.0469	2.2188	1.3281

Table 2: Average CPU Time for Finding Defective Cliques in sec

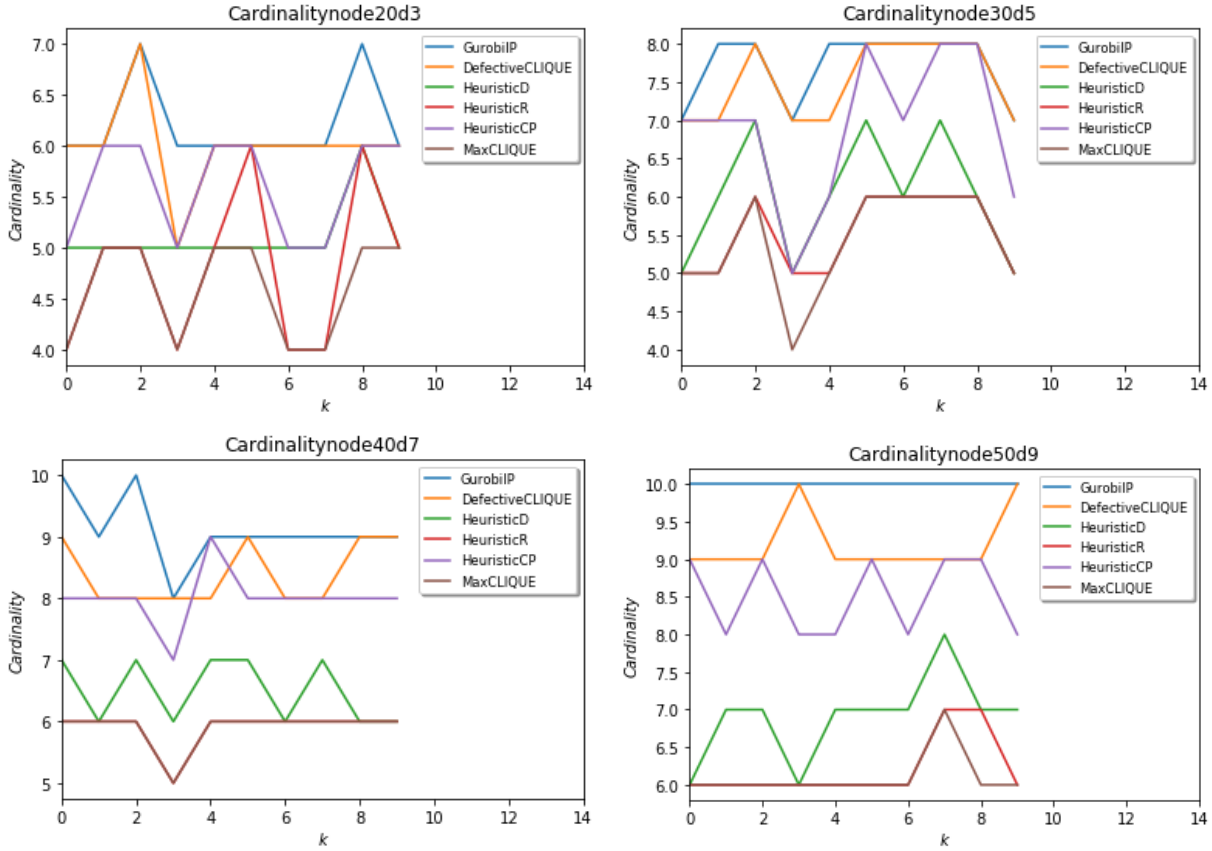

Figure 1: Cardinality of Defective Cliques for ER Graph

6 Analysis of Results

6.1 Quality of Solution Obtained

We notice from the results in the previous section that the proposed ODCliques algorithm performs very well and produces results almost similar to the s-defective IP formulation done in Gurobi. We also see that the C_p heuristic algorithm also performes relatively well among the other two heuristic algorithms, with Degree-Rank outperforming Random. All of them produce defective cliques of equal or higher cardinality than the maximum clique, hence validating that they all produce feasible solutions.

6.2 Computation Time Comparison

We notice from the results that the proposed OD-Cliques algorithms performs better on average than the s-defective IP Forumulation in Gurobi, while generating results of similar quality. However the computation time used by the C_p heuristic, is extremely low in comparison with these two and difference becomes more stark as the number of nodes of the graph G rises. This heuristic provides a solution of very good quality at a very cheap cost.

6.3 Scope for Improvement

We noticed certain further modifications that may be implemented in our algorithm to make it more

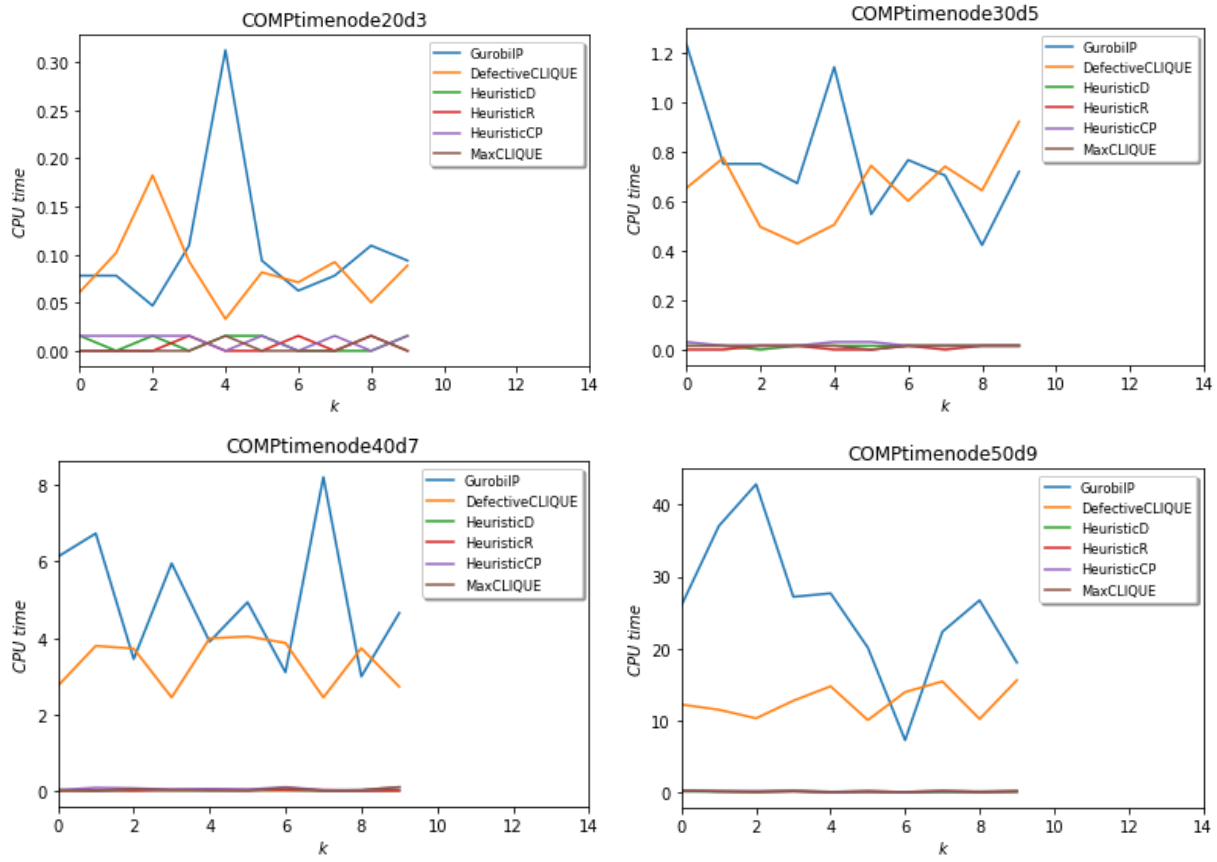

Figure 2: CPU Times of Max Defective Clique Algorithms

efficient as well as generate more possibilities of identifying higher cardinality defective cliques.

- We can further prune several branches of the DFS tree in order to prevent useless computation. For instance, while considering growing cliques of size $\omega_G - s$ where ω_G is the size of the maximum clique of the graph, G, we see that it can only outperform the maximum clique if every node only accounts for one defect. Hence, if we find any node addition candidate which adds more than one defect, we can immediately stop continuing the DFS in that direction.

- While iterating over the set **CAND-n**, the elements of this set are not sorted according to their highest degree in the induced subgraph. This sorting will ensure that those nodes which contribute minimum defects to the already existing clique or defective clique are selected first. This can also help us prune the DFS tree since if adding an earlier node in this set cannot result in a better clique than one already achieved, we can stop the search in that direction.

- While selecting the first node of the DFS, we are only considering the neighbors of it as candidates for forming the clique as well as prospective s-defective cliques. However, there might be some nodes which are not connected to the first node of the DFS, but very well connected to its neighbors, owing to which it has an opportunity to be a part of the defective clique, if there are defects left to be filled at the end of the DFS. But the initial pruning will prevent those nodes from being part of the defective clique. This is a prospective modification scope for our algorithm.

7 Conclusions

Here we see that the cardinality of the maximum s-defective clique provided by the ODCLIQUES algorithm is in many instances equal to that provided by the commercial solver Gurobi 9.0. Our algorithm by design offers the advantage that it lists a set of defective cliques and we can leverage that information to easily combine elements of defective cliques of the highest sizes to create an even larger

defective clique in a post-processing stage.

7.1 Contribution of Team Members

We have both contributed in develop and implement the algorithm and taken turns in the debugging procedure.

References

[1] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011.

[2] R Duncan Luce and Albert D Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[3] R Duncan Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, 15(2):169–190, 1950.

[4] Richard D Alba. A graph-theoretic definition of a sociometric clique. *Journal of Mathematical Sociology*, 3(1):113–126, 1973.

[5] Robert J Mokken et al. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979.

[6] Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[7] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.

[8] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006.

[9] Mihalis Yannakakis. Node-and edge-deletion np-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264, 1978.

[10] Oleg A Shirokikh. *Degree-based Clique Relaxations: Theoretical Bounds, Computational Issues, and Applications*. PhD thesis, University of Florida, 2013.

[11] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science*, 363(1):28–42, 2006.

[12] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.