Sabanci University

Fall 2019-2020

CS301 - Algorithms

Project Report

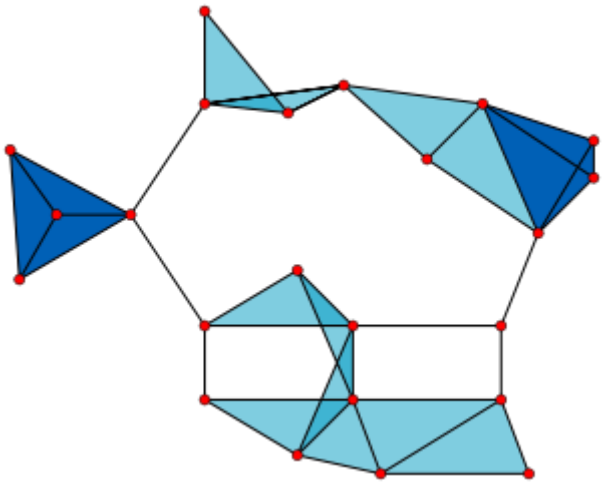
k-Clique

Alper Giray, Furkan Özhan, Onat Kutlu, Onur Arda Bodur, Ömer Sezer Koyuncu

Problem Description:

Clique as a word means a small group of people, with shared interests or other features in common, who spend time together and do not readily allow others to join them.

In the context of computer science, clique is a subset of vertices in an undirected graph that has all the vertices adjacent to each other which means they are all connected by edges to each other.



Clique problem is the computational problem of finding cliques. Based on the need and given parameters, there may be different formulations in the problem.

Common formulations of the clique problem include finding a maximum clique which is a clique with the largest possible number of vertices, finding a maximum weight clique in a weighted graph, or listing all maximal cliques that are cliques that cannot be enlarged.

In this project's context, the problem specified is finding a clique with a requested size. It is described more formally below:

Input: *We have an n-node undirected graph G (V, E) with node set V and edge set E; a positive integer k with* $k \leq n$.

Problem: *Does G contain a k-clique, i.e. a subset W of the nodes V such that W has size k and for each distinct pair of nodes u, v in W, {u, v} is an edge of G?*

Clique computation is used in various fields, with bioinformatics being the most common. Other than modeling in bioinformatics, it is also used in electrical engineering and chemistry fields.

In bioinformatics, the problem of clustering gene expression data requires finding the minimum number of changes needed to transform a graph describing the data into a graph formed as the disjoint union of cliques.

In electrical engineering, cliques are used to analyze communications networks, and to design efficient circuits for computing partially specified Boolean functions.

In chemistry, cliques are used to describe chemicals in a chemical database that have a high degree of similarity with a target structure.

Most versions of the clique problem are hard. The clique decision problem is NP-complete (one of Karp's 21 NP-complete problems).

The problem of finding the maximum clique is both fixed-parameter intractable and hard to approximate. And, listing all maximal cliques may require exponential time as there exist graphs with exponentially many maximal cliques.

k-Clique is NP-Complete.

Proof:

In order to prove a problem is NP-Complete:

- Problem needs to be shown in NP.
- It needs to be reduced to a NP-Complete problem in polynomial time.

1. To show CLIQUE is in NP, our verifier takes a graph G(V,E), k, and a set S and checks if $|S| \geq k$ then checks whether $(u, v) \in E$ for every $u, v \in S$. Thus the verification is done in $O(n^2)$ time.

2. Next we need to show that CLIQUE is NP-hard; that is we need to show that CLIQUE is at least as hard any other problem in NP. To do so, we give a reduction from 3-SAT (which we've shown is NP-complete) to CLIQUE. Our goal is the following:

Given an instance φ of 3-SAT, we will produce a graph G(V,E) and an integer k such that G has a clique of size at least k if and only if φ is satisfiable.

Let φ be a 3-SAT instance and $C_1, C_2, ..., C_m$ be the clauses of φ defined over the variables $\{x_1, x_2, ..., x_n\}$. What we need to do is construct an instance of CLIQUE (a graph) that would somehow capture the satisfiability of the clauses of φ.

We will represent every clause C_i as $C_i = \{z_{i1}, z_{i2}, ..., z_{it}\}$ where each z_{ij} represents a literal in C_i. Since φ is a 3-SAT instance, we know that $t \leq 3$.

We construct a graph G(V,E) by adding t vertices for every clause $C_i = \{z_{i1}, z_{i2}, ..., z_{it}\}$. In total this takes $O(t \cdot m) = O(m)$ time since $t \leq 3$. Then for every pair of vertices v_{ab}, v_{cd} in G, we will add the edge (v_{ab}, v_{cd}) if and only if we satisfy two conditions:

$$a \neq c \quad (1)$$

$$Z^{ab} \neq \neg Zcd \quad (2)$$

What do these two conditions mean? Well (1) implies that the literals z_{ab}, z_{cd} corresponding to the vertices v_{ab}, v_{cd} respectively belong to different clauses $C_a = C_c$ in φ. The second condition implies that both literals can be satisfied simultaneously. This step of the construction takes $O(m^2)$ time. The final step is to determine the value of k; we will set k to be m, the number of clauses in φ.

Now I claim that φ is satisfiable if and only G as constructed above has a clique of size at least $k = m$.

First suppose φ is satisfiable. Then there exists a satisfying assignment $(x^*_1, x^*_2, ..., x^*_n)$ such that every clause C_i in φ is satisfied. Notice that to satisfy a clause C_i , we just need

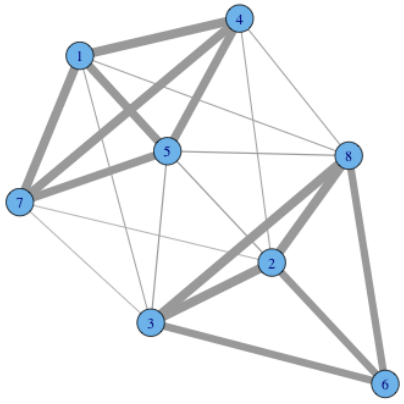
one of its literals in $\{z_{i1}, z_{i2}, ..., z_{ik}\}$ to be satisfied. We iterate through the clauses and choose one satisfied literal from every clause which we denote by $(z_{*1}, z_{*2}, ..., z_{*m})$. Let $v_1, v_2, ..., v_m$ be the corresponding vertices in G to the satisfied literals we selected.

The set $S = \{v_1, v_2, ..., v_m\}$ must form an m-clique in G. Why? Well notice that $z_{*1}, z_{*2}, ..., z_{*m}$ all have the same truth assignment, since otherwise $z*i = \neg z*j$ for some $i, j \in [1,m]$ thus implying that one of $z*i$ and $z*j$ is not the satisfying literal of C_i, C_j, a contradiction to our choice of the $z*$ literals. Notice also that the z_*'s belong to different clauses, that's how we chose them. Therefore, by the construction of G, every pair of $v_1, v_2, ..., v_n$ must have a connecting edge and thus $S = \{v_1, v_2, ..., v_m\}$ forms an m-clique in G.

Conversely, suppose G has a clique of size at least $m = k$. Let $v_1, v_2, ..., v_q$ be a clique in G of size $q \geq m$, then the first m vertices $v_1, v_2, ..., v_m$ must also form a clique in G. Since there are no edges connecting vertices from the same clause, every v_i corresponds to a literal zi from exactly one clause C_i. Moreover, since $v_1, v_2, ..., v_m$ is a clique, the corresponding literals zi,zj of any pair $vi,vj \in \{v_1, v_2, ..., v_m\}$ can be satisfied simultaneously (by construction). Now, to construct a satisfying assignment x1, ..., xn for φ, we just need to satisfy all of z1, ..., zm and assign the remaining variables arbitrarily. Every C_i contains one z_i, and every z_i is satisfied thus every C_i is satisfied and so φ is satisfied.

To conclude, we've shown that CLIQUE is in NP and that it is NP-hard by giving a reduction from 3-SAT.

Therefore, CLIQUE is NP-complete.



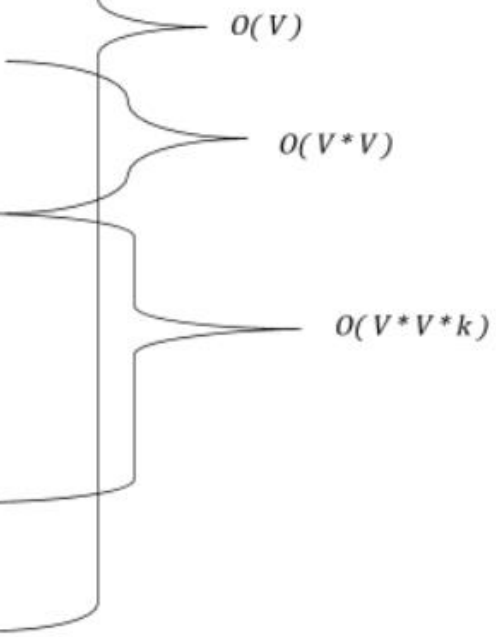
Algorithm Description:

There is not an exact algorithm to solve the k-Clique problem but there are some possible heuristics that can be implemented such as the greedy algorithm of Grimmett-McDiarmid's to find cliques which works with a high probability but still there can be errors due to all possible vertex can be form different cliques with all its adjacent. Our algorithm approaches in a greedy way. Basically, the greedy approach is if there is an adjacency it must be in a clique relationship with given vertex. However, the error occurs that if that adjacent vertex cannot form a clique with size k, then the algorithm gives an erroneous result. But with this approach of not considering all cliques of a given graph, the algorithm finds the solution in a shorter time than Non-Polynomial algorithms.

```
def p_clique(graph,k):
  if k == 1:
      return True
  clique = []
  vertices = get_keys_array(graph)
  max_ln=0
  max_dict=[]
  for i in range(0,len(graph)):
    clique=[]
    clique.append(vertices[i])
    for v in vertices:
        if v in clique:
            continue
        isNext = True
        for u in clique:
            if u in graph[v]:
              continue
            else:
              isNext = False
              break
        if isNext:
          clique.append(v)
          if k <= len(clique):
          return True
    if len(clique)>max_ln:
      max_ln=len(clique)
      max_dict=clique
  if k <= len(clique):
          return True
  else:
          return False
```

Algorithm Analysis:

```
def p_clique(graph,k):
  if k == 1:
      return True
  clique = []
  vertices =
get_keys_array(graph)
  max_ln=0
  max_dict={}
  for i in range(0,len(graph)):
    clique=[]
    clique.append(vertices[i])
    for v in vertices:
        if v in clique:
            continue
        isNext = True
        for u in clique:
            if u in graph[v]:
              continue
            else:
              isNext = False
              break
        if isNext:
          clique.append(v)
          if k <= len(clique):
            return True
    if len(clique)>max_ln:
      max_ln=len(clique)
      max_dict=clique
  if k <= len(clique):
            return True
  else :
            return False
```



Running Time: $O(V^2 + k)$

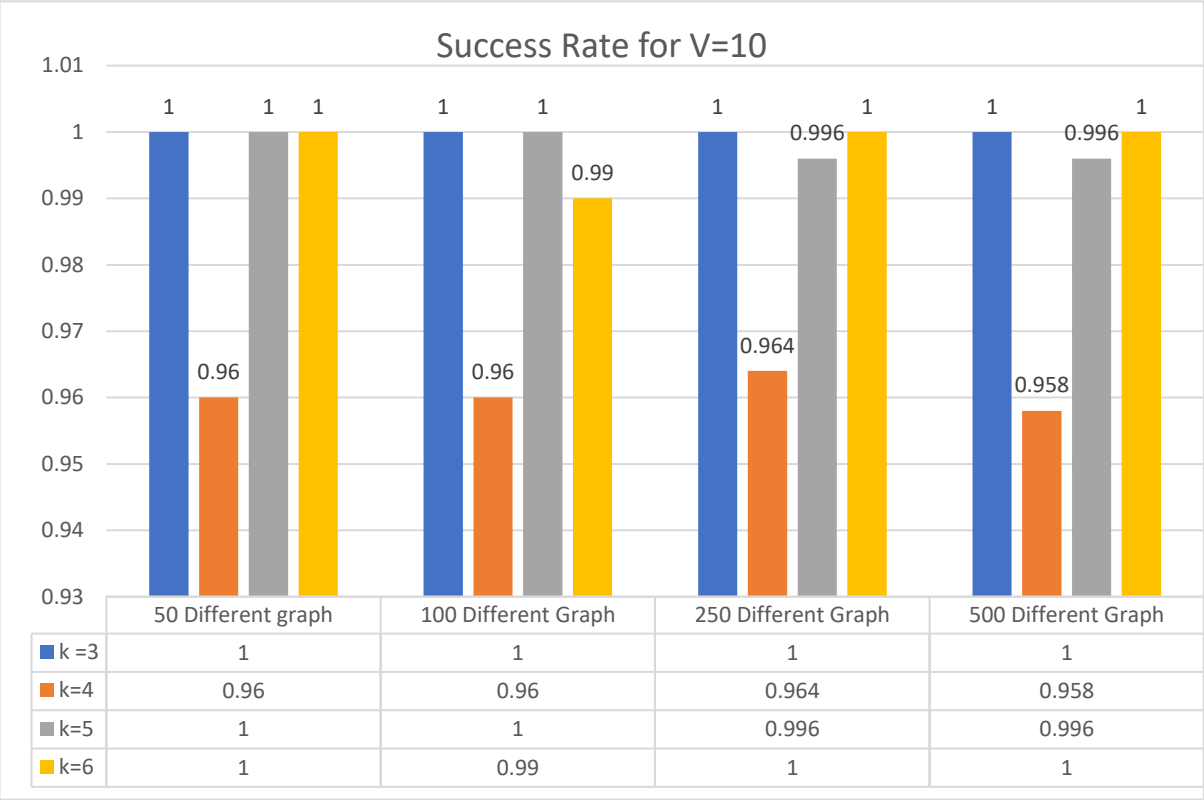
The algorithm we have supplied does not guarantee computing without errors because it doesn't consider every adjacent vertex for given vertex and it tries to find just one clique with a given(k) sized for each vertex. The worst case running time of the algorithm is calculated as stated above.

Experimental Analysis

We tested the algorithm with randomly generated graphs of sizes 10, 15, 20, 30 with k values of 3, 4, 5, 6 for 50, 100, 250, 500 times in sequence.

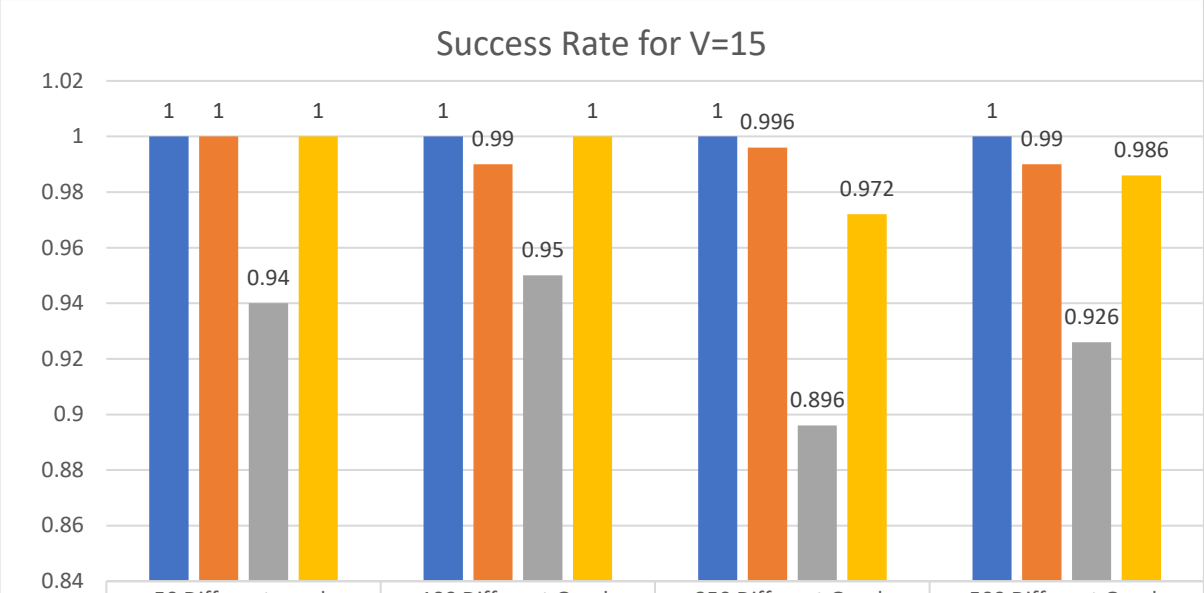
Succes Rates

Success Rates are determined as if our algorithm can find k-clique precisely.



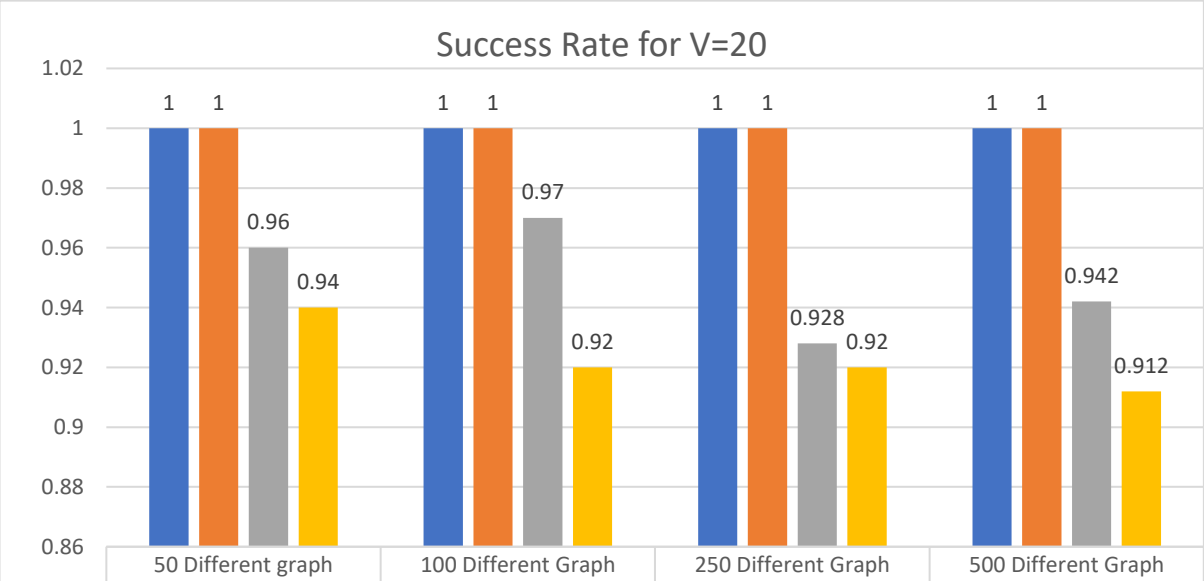
	50 Different graph	100 Different Graph	250 Different Graph	500 Different Graph
k =3	1	1	1	1
k=4	0.96	0.96	0.964	0.958
k=5	1	1	0.996	0.996
k=6	1	0.99	1	1

Table 1. Success Rates table and graph of randomly generated graphs with 10 vertices and different k values.



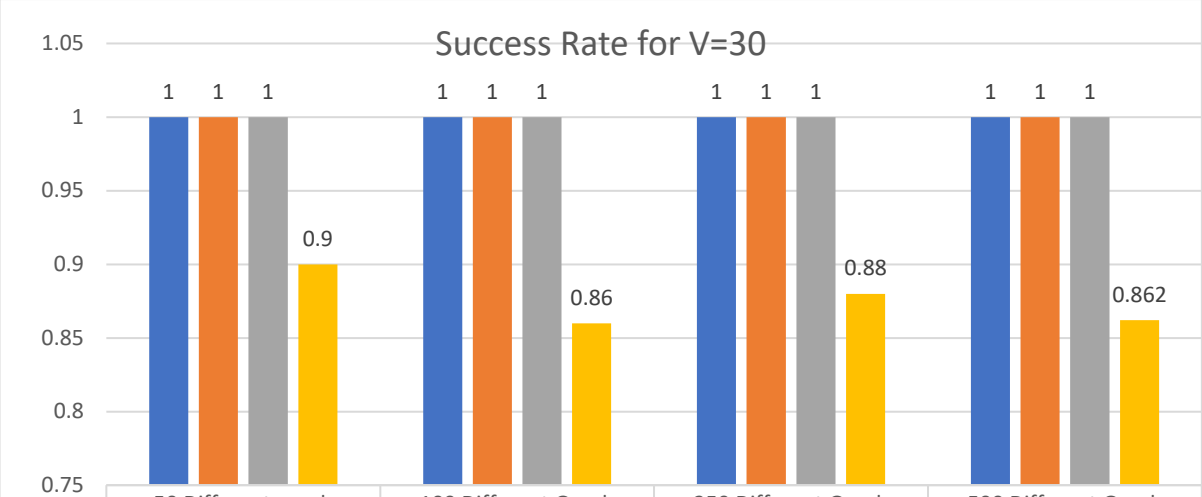
	50 Different graph	100 Different Graph	250 Different Graph	500 Different Graph
k =3	1	1	1	1
k=4	1	0.99	0.996	0.99
k=5	0.94	0.95	0.896	0.926
k=6	1	1	0.972	0.986

Table 2. Success Rates table and graph of randomly generated graphs with 15 vertices and different k values.



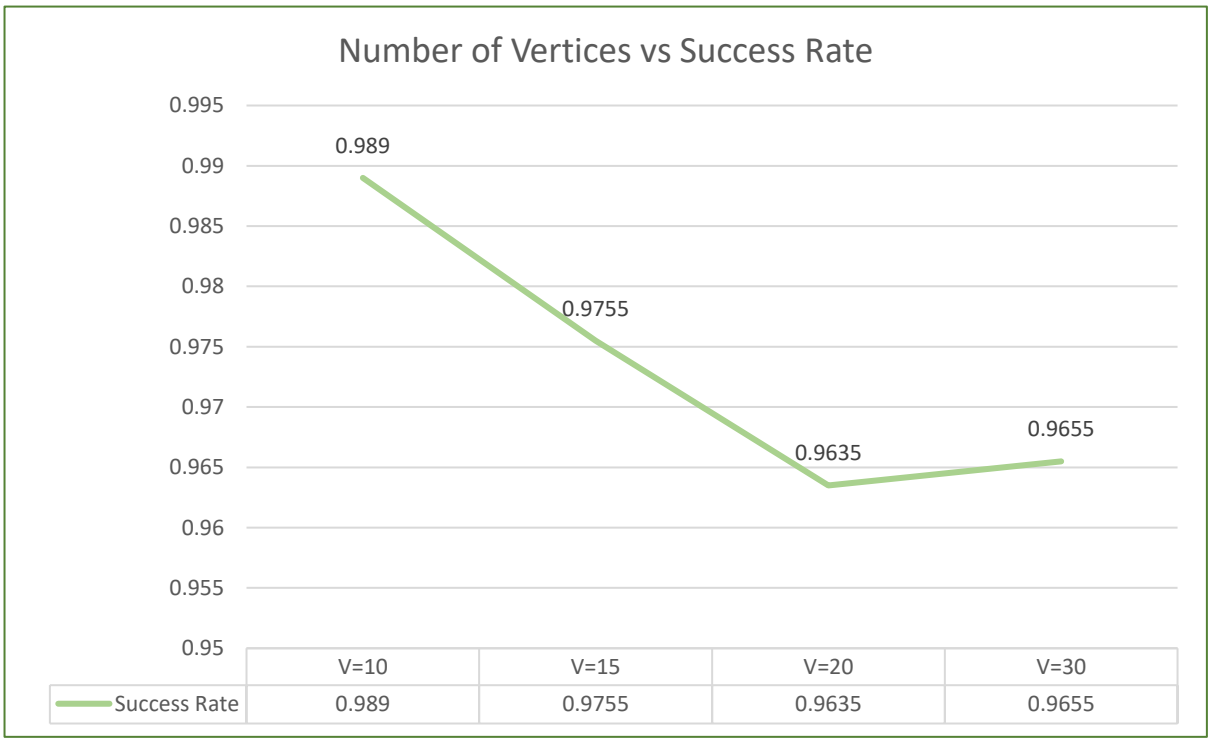
	50 Different graph	100 Different Graph	250 Different Graph	500 Different Graph
k =3	1	1	1	1
k=4	1	1	1	1
k=5	0.96	0.97	0.928	0.942
k=6	0.94	0.92	0.92	0.912

Table 3. Success Rates table and graph of randomly generated graphs with 20 vertices and different k values.



	50 Different graph	100 Different Graph	250 Different Graph	500 Different Graph
k =3	1	1	1	1
k=4	1	1	1	1
k=5	1	1	1	1
k=6	0.9	0.86	0.88	0.862

Table 4. Success Rates table and graph of randomly generated graphs with 30 vertices and different k values.



	V=10	V=15	V=20	V=30
Success Rate	0.989	0.9755	0.9635	0.9655

Graph 6. Graph of randomly generated graphs with different vertices

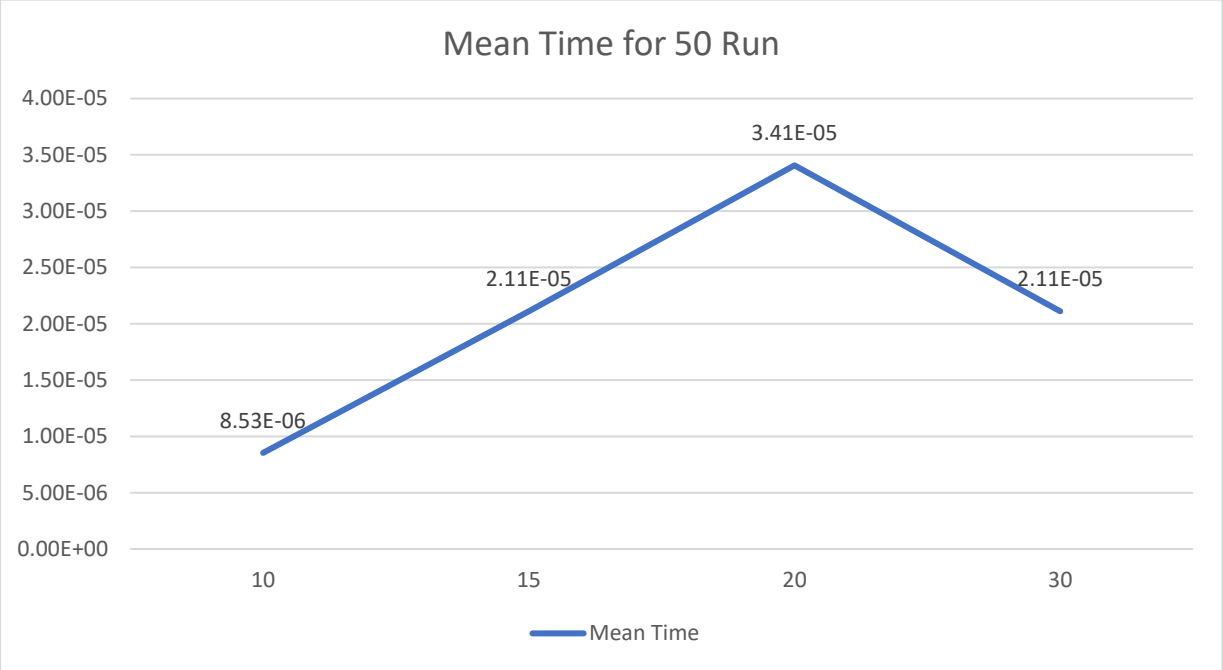
*It can be understood that while number of vertices are increasing, accuracy of our problem increases. However, reason of that our clique sizes become smaller compared to graph size and the occurrence of k-cliques and our greedy algorithm can find cliques easily.

Running Time Experimental Measurement

To experimentally analyze runnig time of our algortihm we created some functions to help us to interpret some statistic concepts using python libraries. We fixed our K value to 5 because we want to find the effect of only the number of vertices to running time.

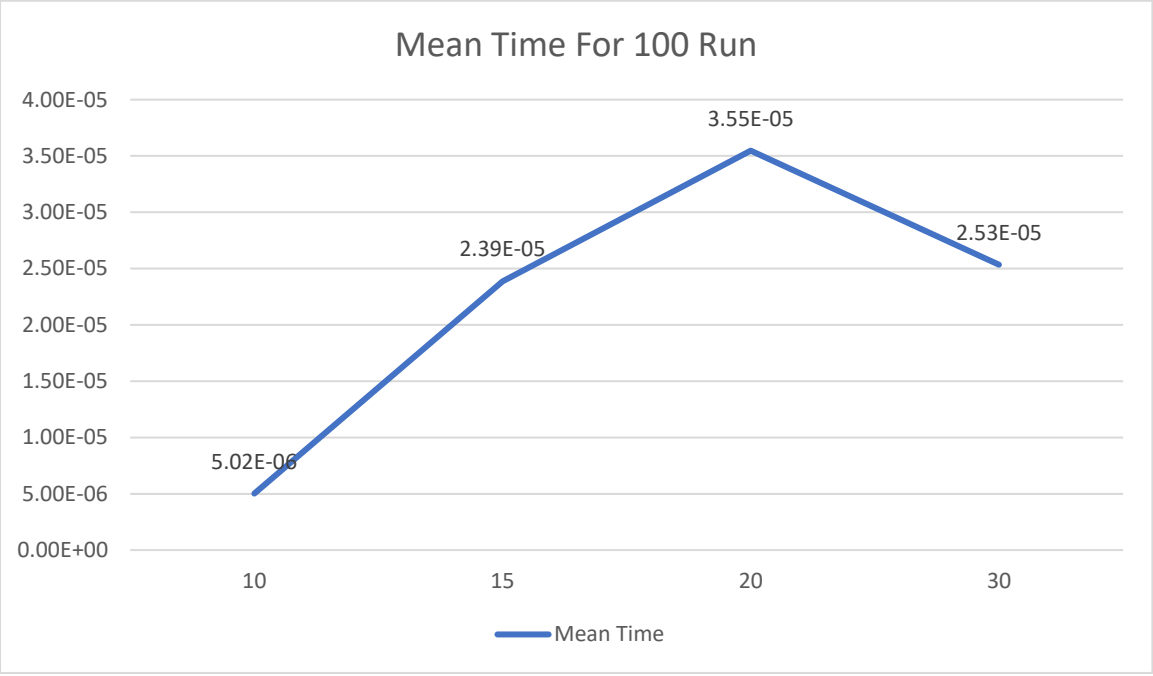
50 Run Per Input Size

Size	Mean Time	Standard Deviation	Standard Error	%90-CL	%95-CL
10	8.534e-06	2.75e-05	1.21e-06	2.95e-05 - 2.55e-05	2.99e-05 - 2.51e-05
15	2.110e-05	3.507e-05	3.015e-06	4.003e-05 - 3.011e-05	4.098e-05 - 2.9166e-05
20	3.407e-05,	4.050e-05	4.867e-06	4.851e-05 - 3.249e-05	5.004e-05 - 3.096e-05
30	2.111e-05	2.256e-05	3.016e-06	2.752e-05 - 1.760e-05	2.847e-05 - 1.665e-05



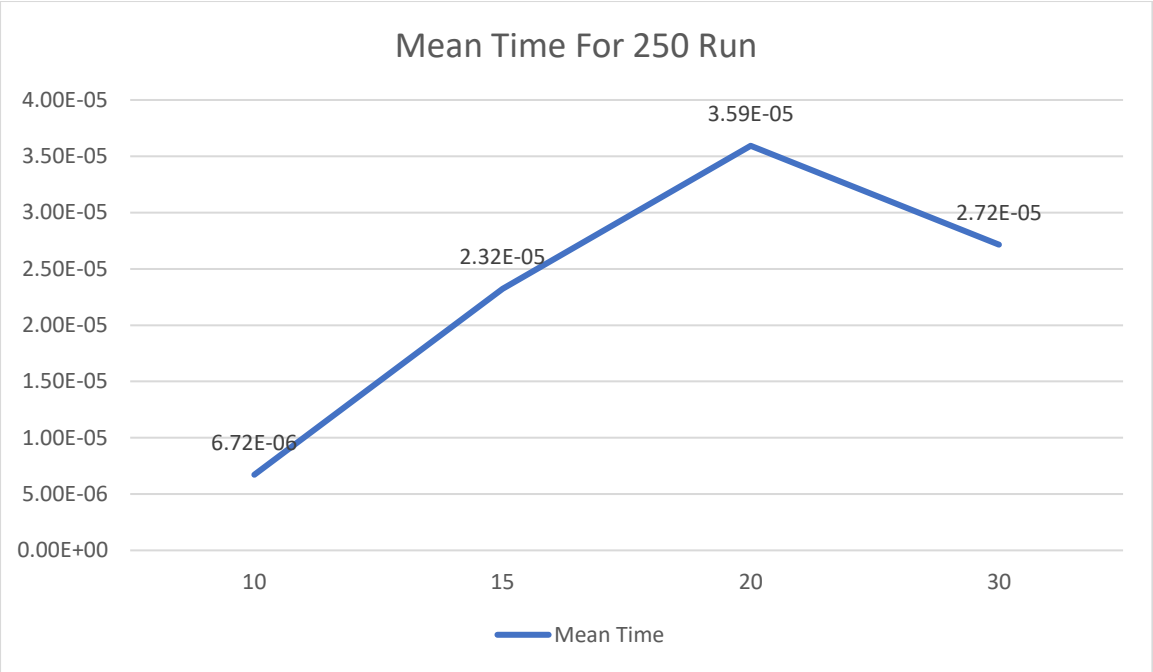
100 Run Per Input Size

Size	Mean Time	Standard Deviation	Standard Error	%90-CL	%95-CL
10	5.016e-06	2.57e-05	5.04e-07	2.65e-05 - 2.49e-05	2.673e-05 - 2.476e-05
15	2.385e-05	3.368e-05	2.397e-06	3.7625e-05 - 2.9737e-05	3.838e-05 - 2.898e-05
20	3.547e-05	4.064e-05	3.564e-06	4.650e-05 - 3.478e-05	4.763e-05 - 3.365e-05
30	2.533e-05	2.420e-05	2.546e-06	2.839e-05 - 2.001e-05	2.919e-05 - 1.921e-05



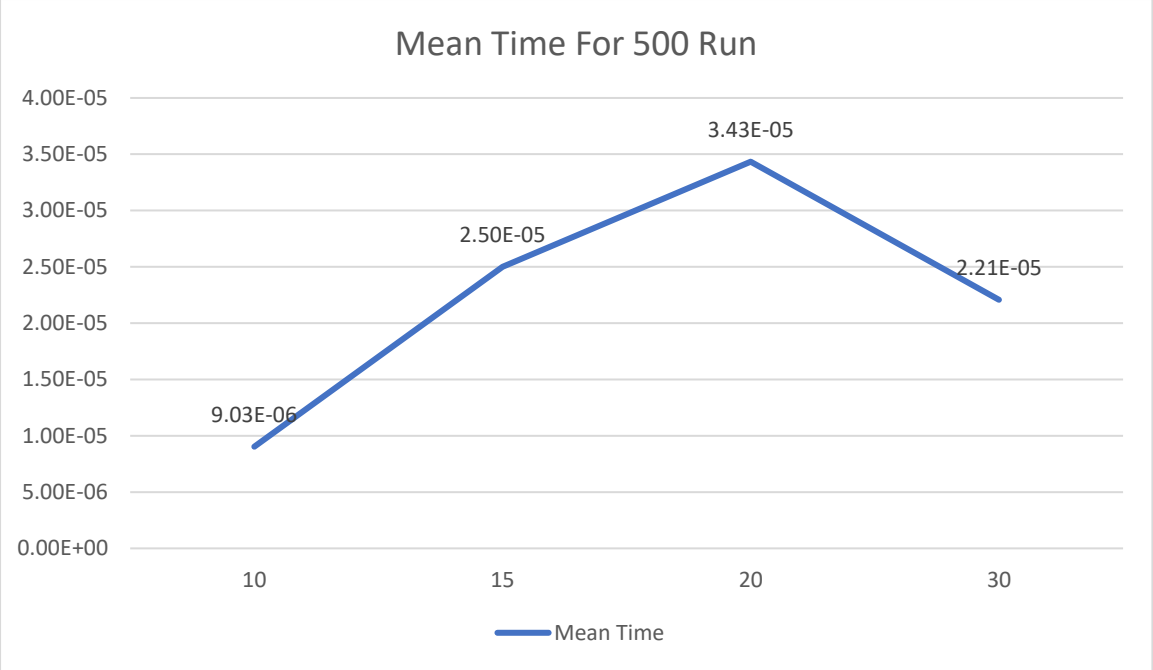
250 Run Per Input Size

Size	Mean Time	Standard Deviation	Standard Error	%90-CL	%95-CL
10	6.718e-06	2.765e-05	4.257e-07	2.835e-05 - 2.695e-05	2.848e-05 - 2.682e-05
15	2.323e-05	3.928e-05	1.472e-06	4.170e-05 - 3.686e-05	4.216e-05 - 3.639e-05
20	3.593e-05	3.887e-05	2.277e-06	4.262e-05 - 3.513e-05	4.333e-05 - 3.441e-05
30	2.715e-05	2.780e-05	1.720e-06	3.063e-05 - 2.497e-05	3.117e-05 - 2.443e-05



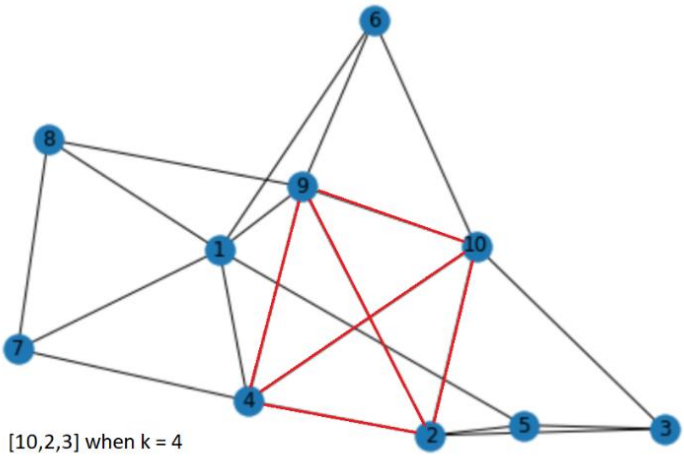
500 Run Per Input Size

Size	Mean Time	Standard Deviation	Standard Error	%90-CL	%95-CL
10	9.031e-06	2.753e-05	4.042e-07	2.820e-05 - 2.687e-05	2.833e-05 - 2.674e-05
15	2.502e-05	4.352e-05	1.120e-06	4.537e-05 - 4.168e-05	4.572e-05 - 4.133e-05
20	3.434e-05	3.734e-05	1.537e-06	3.987e-05 - 3.481e-05	4.035e-05 - 3.432e-05
30	2.208e-05	2.581e-05	9.885e-07	2.744e-05 - 2.419e-05	2.775e-05 - 2.387e-05

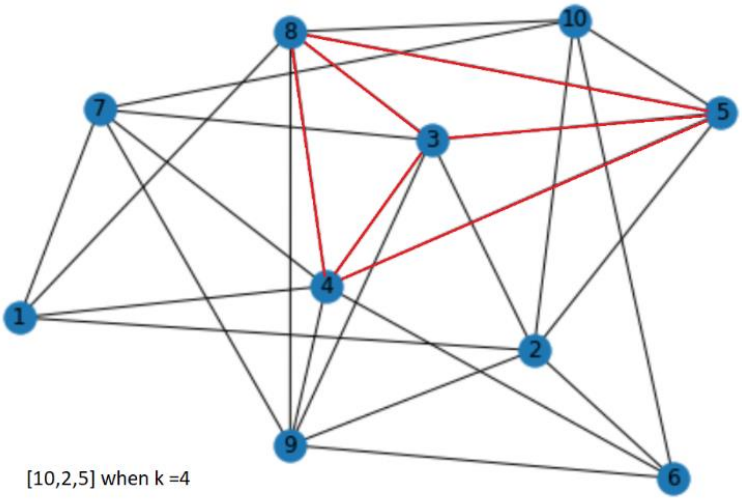


TESTING

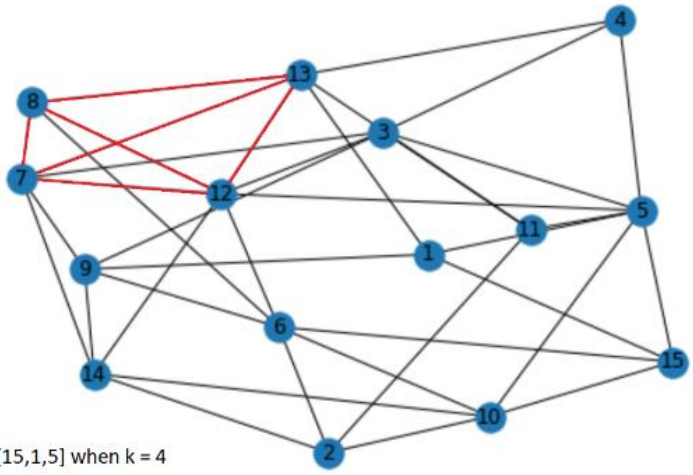
Failed Conditions



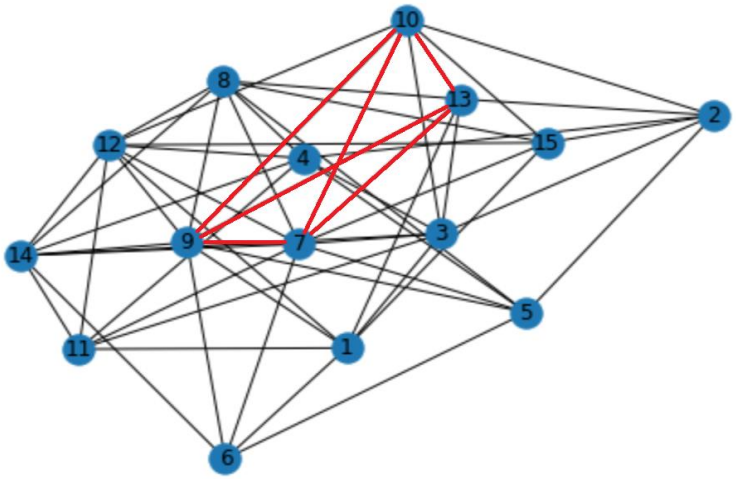
[10,2,3] when k = 4



[10,2,5] when k =4



[15,1,5] when k = 4



Successful Case

When k = 4 both algorithm returns true(p algo = [7,3,10,13])
When k = 5 p algorithm returns false but np algo finds =[7,9,8,12,14]

CONCLUSION

To sum up, we have seen that finding k-clique problem is NP-Complete problem that is reduced from 3-SAT. Also, there is not such an exact efficient algorithm that solves that problem. There are just heuristic algorithms such as greedy max clique algorithms. We changed this algortihm to find k-clique. We examined and analyzed this algorithm in this report and have seen that this algortihm does not always work.

After experimental analysis of this algorithm, we deduced that it has aproximately 100% chance to find answer for decision problem.

If we consider the measurement of runnig time of the algorithm in the experimental analysis part shows a consistent trend with our result. However, when the graph size is 30, the running time decreased because when the graph size increases, number of connections are also increased so finding 5-clique is become easier and fast. Then it is normal to algorithm become faster.