CS 301 – Algorithms

Fall 2021-22

Group 38 Project Final Report

1. Problem Description

Finding the minimum number k such that the nodes of G be partitioned into t <= k disjoint sets - V1, ..., Vt - in such a way that for each Vi (1 <= i <= t), the subgraph Gi (Vi, Ei) induced by Vi is a clique, meanly a graph in which every pair of nodes is connected by an edge.

Given a graph, our aim is to divide the graph into a minimum number of parts such a way that each divided part is a clique which means every node is connected to each other by an edge. This minimum number is represented by “k”.

The problem of deciding whether the edge-set of a given graph can be partitioned into at most k cliques is well known to be NP-complete. (Egbert Mujini et. al., 2008)1

In computational complexity theory, a problem is NP-Complete when it is a problem for which the correctness of each solution can be verified quickly and a brute-force search algorithm can actually find a solution by trying all possible solutions. The theory of NP-completeness is designed to

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

be applied only to decision problems. Such problems have only two possible solutions, either the answer “yes” or the answer “no”. The reason for restriction to decision problems is that they have a very natural, formal counterpart, which is suitable to study in a mathematically precise theory of computation. Problem of finding the minimum number of cliques which partition or cover the edge-set of a graph, can be derived to a decision problem in the following way.

CP(G,k)

INSTANCE: A graph G = (V, E) and an integer k

QUESTION: Is there a partition of E into no more than k cliques ?

Now if we have a “good” algorithm for finding cp(G) for any graph G, then we could use this algorithm to solve the decision problem. The class NP is also defined informally to be the class of all decision problems that can be solved by polynomial time nondeterministic algorithms. (Sebastian M. Cioaba, 2005)2 Due to the fact that our algorithm’s complexity is in polynomial time, and it can be easily derived to a decision problem, it can be said that it is NP-complete.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

2. Algorithm Description

a.

1. Implement the graph using an adjacency matrix and call it A[i, j], i and j represents vertex numbers and the value of A[i, j] represents edges. For example, if A[i, j] = 1 then there exists an edge between two vertices i and j.

2. Create a list of vertices and initialize N as length of the list

3. Initialize k as 1

4. Create a list that contains k subset lists dynamically

5. Try every combination of subsets with recursion until there are no nodes left in the vertices list.

6. Check if every subset in the subsets list represents a clique by checking if every pair of nodes is connected by an edge.

7.If it is true, we found our k number, go to step 9.

8. If it is false, increment the k value. Go to step 4.

9. Print the k value

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

This algorithm has exponential time complexity depending on k and n values as k^n. So, it is an time-inefficient algorithm that only can solve the small problems.

```python
import copy
def checkClique(M,subsets):
  clique = True
  for subset in subsets:
    #print('a')
    for element in subset:# [1]
      for other in subset:
        if element != other and M[element][other] == 0 :
          clique = False
  return clique

def Partition(S,N,subsets,k,M):
  if k > len(S):
     print('k is larger than number of vertices')
     return False
  if k == len(S):
    result= []
    for element in S:
      result.append([element])
    print('partitioned with k =', k ,' as ', '1-'*(k-1)+'1')
    print(result)
    return True
  # if everything is ok start recursion []
  for i in range(len(subsets)): #[6,5,3,2,1,0][][]
    subsetsRec = copy.deepcopy(subsets)
    if N>0:
      subsetsRec[i].append(S[N-1])
      if Partition(S,N-1,subsetsRec,k,M):
        return True
      #print(subsetsRec)
    else:
      #print('N bitti', subsetsRec)
      if checkClique(M,subsetsRec):
        print(subsetsRec)
        print('partitioned with k =', k , 'as',[len(subsetsRec[i]) for i in range(k)])
        return True;
Set = [i for i in range(length)]
N = len(Set)
k =1
found = False

while not found:
  subsets = [[] for i in range(k)]
  found = Partition(Set,N,subsets,k,A)
  if found : print("Our k value is =",k)
  k+=1

```

b.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

1. Implement the graph using an adjacency matrix and call it A[i, j], i and j represents vertex numbers and the value of A[i, j] represents edges. For example, if A[i, j] = 1 then there exists an edge between two vertices i and j.
2. Initialize an empty list to keep track of clique. Call it C.
3. Initialize an empty integer list to keep track of the number of cliques in the graph, call it cliqueList.
4. Initialize a variable to keep track of minimum k value, call it maxSize as 0.
5. A while loop, iterates till there are no vertices left in the graph.
6. Initialize a variable to keep track of the number of divisions, call it total.
7. Two for loops which are called tempi and tempj are used to iterate and find all possible starting paths of cliques.
8. Inside the tempi for loop, initialize one more loop which checks the single vertex cliques and add those vertices to cliqueList.
9. Initialize N1 as i.
10. Initialize j as tempj.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

11. For each starting path, the loop initially checks whether C is empty and if it is empty adds the initial vertices, which corresponds to our very first clique.
12. For each starting path, a while loop checks whether that particular node called j is already in C or not. If j is already in our list C, the loop terminates. If it is not, it checks if there is an edge between j and all the vertices in the list C and the ones with an existing edge. In order to pass to the next vertex, assign j equals to i and j equals to 0.
13. After the while loop is terminated, if the size of C is greater than maxSize, update maxSize as the size of C.
14. Remove the vertices from the graph which are the ones composing the biggest clique in order to check if the remaining vertices create another clique or not.
15. Add size of C to the cliqueList.
16. The value of 'total' is the minimum number of cliques in the graph, k, which we are looking for.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

3. Algorithm Analysis

a.

In our brute force algorithm, code starts with a while loop that runs until we find an k value that satisfies the condition of our problem. In this while loop, we create subsets list that contain k empty subset lists dynamically. We call the partition function with k and n values. Then, assign the return of the function to the found variable. After that we increment the k value. If we found the k value, then the while loop terminates and prints the k value. Partition function works with the recursion manner. And the number of recursions in one call is equal to k. Recursion is called with n-1 each time. So, Our complexity is T(N)= k * T(N-1). If we try to find the complexity with a substitution method.

$T(N) = K * T(N-1)$

$T(N) = K* (K* T(N-2)) = K^2 *T(N-2)$

$T(N) = K^3 (T(N-3))$

…..,

$T(N) = K^N * T(N-N)= K^N * T(0)$

$T(0) = 1$

$T(N) = K^N * 1= O(k^N)$

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

Also, if the algorithm can not find a solution with a given k, we call the partition function again after increasing the k by 1.

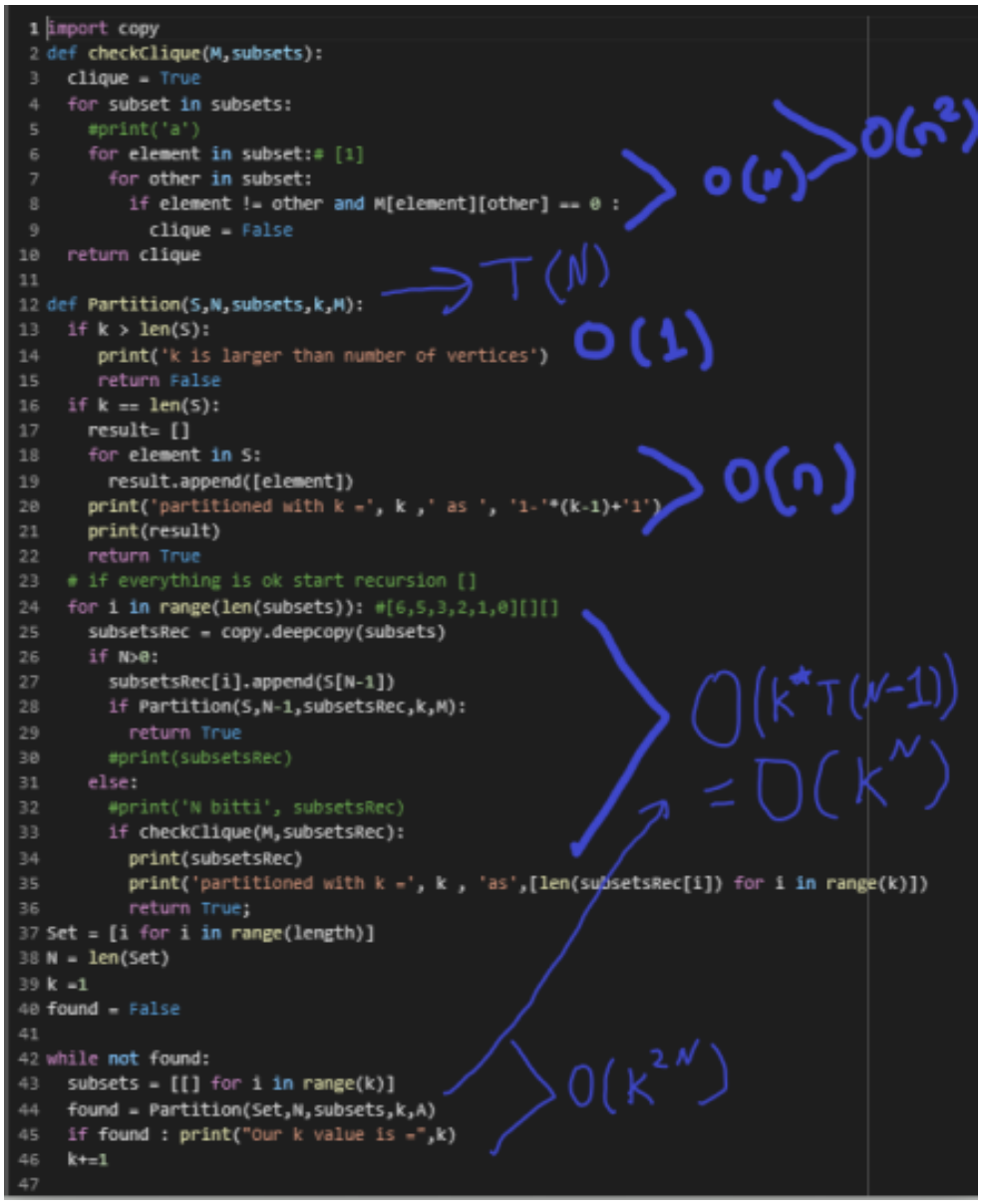
Since we try all the k values, which start from 1, by incrementing by 1 until we find a solution, Algorithm's time complexity is

$O(k^n + (k-1)^n + (k-2)*n + \ldots + 1^n) = O((k * (k+1)/2)^n) =$

$O((k^2)^n) = O(k^{2n})$

This algorithm has exponential running time complexity. Therefore, this is not a time-efficient algorithm and can not converge in large graphs.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf



```python
import copy
def checkClique(M,subsets):
  clique = True
  for subset in subsets:
    #print('a')
    for element in subset:# [1]
      for other in subset:
        if element != other and M[element][other] == 0 :
          clique = False
  return clique

def Partition(S,N,subsets,k,M):
  if k > len(S):
     print('k is larger than number of vertices')
     return False
  if k == len(S):
    result= []
    for element in S:
      result.append([element])
    print('partitioned with k =', k ,' as ', '1-'*(k-1)+'1')
    print(result)
    return True
  # if everything is ok start recursion []
  for i in range(len(subsets)): #[6,5,3,2,1,0][][]
    subsetsRec = copy.deepcopy(subsets)
    if N>0:
      subsetsRec[i].append(S[N-1])
      if Partition(S,N-1,subsetsRec,k,M):
        return True
      #print(subsetsRec)
    else:
      #print('N bitti', subsetsRec)
      if checkClique(M,subsetsRec):
        print(subsetsRec)
        print('partitioned with k =', k , 'as',[len(subsetsRec[i]) for i in range(k)])
        return True;
Set = [i for i in range(length)]
N = len(Set)
k =1
found = False

while not found:
  subsets = [[] for i in range(k)]
  found = Partition(Set,N,subsets,k,A)
  if found : print("Our k value is =",k)
  k+=1
```


b.

In our algorithm, code starts with a while loop whose complexity is O(n) since it iterates till there are no vertices left in the graph. Then it continues with a for loop whose complexity is equal to O(n) since it iterates over the rows of the matrix. Inside that loop, we have two more for loops which are

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

at the same scope, one of them influences the complexity of the algorithm. That loop is for the columns of the matrix and its complexity is O(n) as well. Inside that loop, we have a while loop which checks whether there is an edge between two vertices or not and its complexity is again O(n). Innermost for loop that iterates through the elements of C, and checks whether those elements already exist in C or not. Its complexity is equal to O(k) where k corresponds to the size of our clique.

Inside the while loop, we have some function calls as well, and their complexities are:

```
removeCopies(cliqueList)-->O(n^2)
findMax() --> O(n)
quickSort(max_index,0,len(max_index)-1)--> O(n^2)
removeMax(max_index,A) --> O(n)
```

This part's complexity is equal to O(n^3) + O(n^2) + O(n^3) + O(n^2) = O(n^3). Hence, this part cannot dominate the first part of the while loop. Consequently, after termination of while loop our algorithm's complexity is equal to O(n^4 * k).

4. Sample Generation

1st example,

```
%matplotlib inline
```

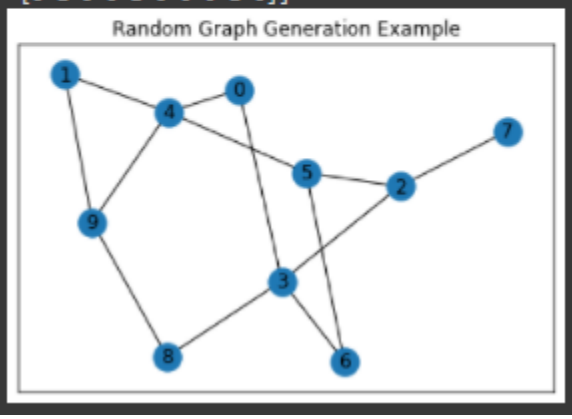

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
from itertools import combinations
from random import random

def ERDetermined(n, p, e):
    V = set([v for v in range(n)])
    E = set()
    remainingE = e
    while remainingE > 0:
      for combination in combinations(V, 2):
          a = random()
          if a < p:
            if combination not in E and remainingE > 0:
              E.add(combination)
              remainingE -=1

    g = nx.Graph()
    g.add_nodes_from(V)
    g.add_edges_from(E)
    array = np.random.randint(0,1, size=(len(V), len(V)))

    for i in E:

      x = i[0]
      y = i[1]
      array[x][y] = 1
      array[y][x] = 1

    print(array)
    return g,array



n = 10
p = 0.3
e = 6
G, A = ERDetermined(n, p, e)
pos = nx.spring_layout(G)
```


1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
nx.draw_networkx(G, pos)
plt.title("Random Graph Generation Example")
plt.show()
```

```
[[0 0 0 1 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 1]
 [0 0 0 1 0 1 0 1 0 0]
 [1 0 1 0 0 0 1 0 1 0]
 [1 1 0 0 0 1 0 0 0 1]
 [0 0 1 0 1 0 1 0 0 0]
 [0 0 0 1 0 1 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 1]
 [0 1 0 0 1 0 0 0 1 0]]
```




```
length = len(A)
length = len(A)
CResultTwos = list()
CResultThrees = list()
CResultFours = list()
C = list()
maxSize = 0
j=0
for tempi in range(length): #O n
  i = tempi #i =1
  N1 = i #n1 = 1
  for  tempj in range(length): #O n
    j = tempj
    while j<length: #O n
```


1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

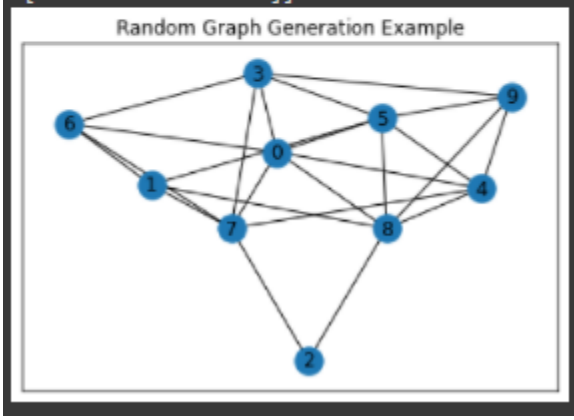
```
      if j in C:
        j+=1
        continue
      elif i == j:
        j+=1
        continue
      elif A[i][j] == 1:
        N2 = j
        if len(C) > 0:
          for element in C: #O K
            if A[element][j] != 1:
              j+=1
              break
            elif element == C[-1]:
              C.append(j)
              i=j
              j=0
              break
        else:
          C.append(N1)
          C.append(N2)
          i = j
          j = 0
      else:
        j+=1
    if maxSize < len(C):
      maxSize = len(C)
    if len(C) > 3 and C not in CResultFours:
      CResultFours.append(C)
    elif len(C) > 2 and len(C)<4 and C not in CResultThrees:
      CResultThrees.append(C)
    elif len(C) > 1 and len(C)<3 and C not in CResultTwos:
      CResultTwos.append(C)
    C = list()
    i=tempi
#print(CResultTwos)
#print(CResultThrees)
#print(CResultFours)
print(maxSize," = k ")
#O (N^3 * K)
```


1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
3 = k
```

2nd example,

```
[[0 0 0 1 1 1 1 1 1 0]
 [0 0 0 0 0 1 1 1 1 0]
 [0 0 0 0 0 0 0 1 1 0]
 [1 0 0 0 0 1 1 1 0 1]
 [1 0 0 0 0 1 0 1 1 1]
 [1 1 0 1 1 0 0 0 1 1]
 [1 1 0 1 0 0 0 1 0 0]
 [1 1 1 1 1 0 1 0 0 0]
 [1 1 1 0 1 1 0 0 0 1]
 [0 0 0 1 1 1 0 0 1 0]]
```

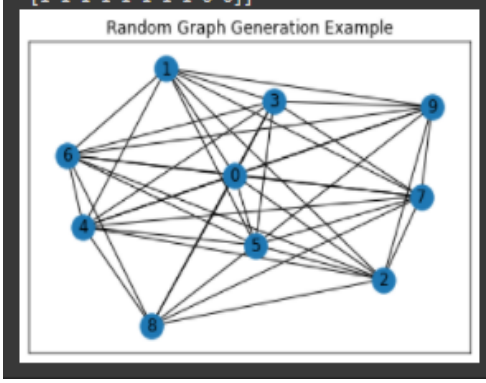



```
4 = k
```

3rd example,

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
[[0 1 1 1 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1 0 1]
 [1 1 0 0 1 1 1 1 1 1]
 [1 1 0 0 1 1 1 1 1 1]
 [1 1 1 1 0 1 1 1 1 1]
 [1 1 1 1 1 0 1 1 1 1]
 [1 1 1 1 1 1 0 1 1 1]
 [1 1 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 1 0 0]
 [1 1 1 1 1 1 1 1 0 0]]
```




```
8  = k
```

5. Experimental Analysis of the Performance (Performance Testing)

```
#performance analysis
max = 5
testRepeat = 100
kListGreedy = []
timeListGreedy = []
standartDevList = []
meanList = []
errorList = []
for n_ in range(max):
 for i in range(testRepeat):
   n = 10 * (n_+1)
   p = 0.2
   e = 40
   G, A = ERDetermined(n, p, e)
   total = 0
   while len(A)>0:  #O n
     total+=1
     cliqueList = list()
     length = len(A)
```


1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
C = list()
maxSize = 0
j=0
for tempi in range(length): #O n
  i = tempi #i =1
  N1 = i #n1 = 1
  alone = True
  for singleVertex in range(length): #O n
    if A[i][singleVertex] == 1:
      alone = False
  if alone:
    C.append(i)
    cliqueList.append(C)
  for  tempj in range(length): #O n
    j = tempj
    while j<length: #O n
      if j in C:
        j+=1
        continue
      elif i == j:
        j+=1
        continue
      elif A[i][j] == 1:
        N2 = j
        if len(C) > 0:
          for element in C: #O K
            if A[element][j] != 1:
              j+=1
              break
            elif element == C[-1]:
              C.append(j)
              i=j
              j=0
              break
        else:
          C.append(N1)
          C.append(N2)
          i = j
          j = 0
      else:
```

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
            j+=1
        if maxSize < len(C):
          maxSize = len(C)
        if len(C) > 0 and C not in cliqueList:
          cliqueList.append(C)
        C = list()
        i=tempi
    #print(maxSize," = k ")
    #O (N^3 * K)
    #print(cliqueList)
    removeCopies(cliqueList)#O n^2
    #print(cliqueList)
    max_index = findMax() #O n
    quickSort(max_index,0,len(max_index)-1)#O n^2
    A = removeMax(max_index,A)#O n
    kListGreedy.append(total)
    timeListGreedy.append((time.time() - start_time))
  print("k = ",total)
    #print("--- %s seconds ---" % (time.time() - start_time))
 standart_deviation = np.std(timeListGreedy)
 standart_error = standart_deviation / math.sqrt(len(timeListGreedy))
 standartDevList.append(standart_deviation)
 errorList.append(standart_error)
 meanList.append(stat.mean(timeListGreedy))
print(kListGreedy)
print(timeListGreedy)
```

The code given in the above figure calculates standard deviation, standard error and mean time in milliseconds for the corresponding running time iterations.

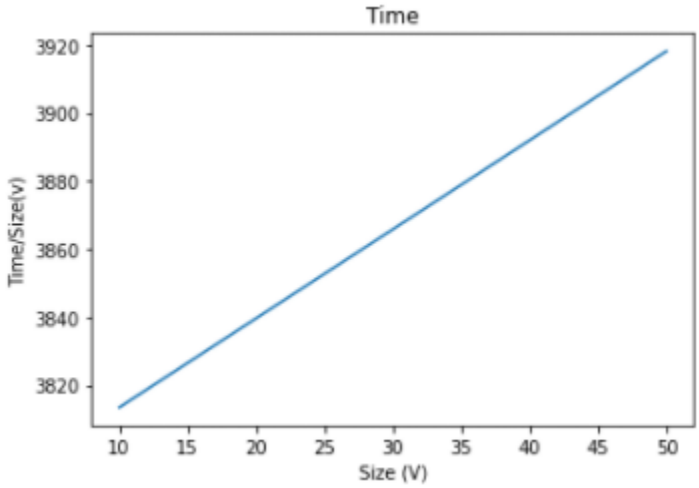
Edge number = 40

N is increasing by 10, starting from 10 till 50.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

size(v)	standart_deviation	standart_error	mean
10	0.234107	0.015239	10581.176587
20	1.011684	0.028300	10582.691313
30	4.714075	0.079276	10587.193924
40	19.079808	0.227933	10603.898733
50	36.362724	0.338232	10629.100578

time: 19.3 ms (started: 2021-12-27 19:31:07 +00:00)

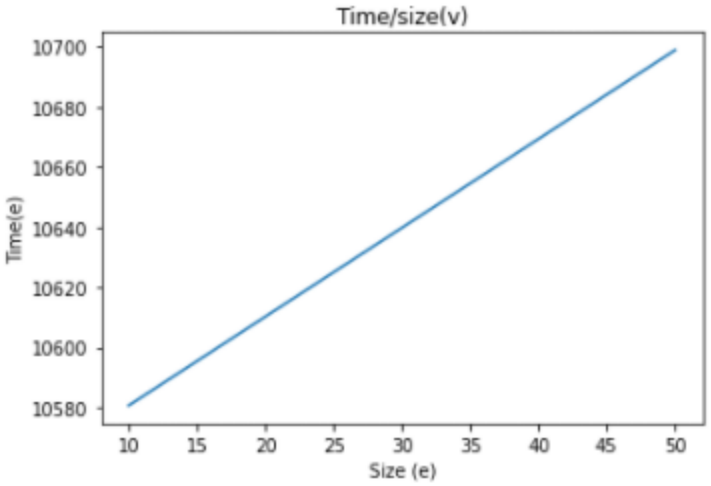


Also, we used a similar algorithm and kept N constant, increased edge numbers respectively.

size(e)	standart_deviation	standart_error	mean
10	0.828112	0.019893	4276.629834
20	1.534015	0.027097	4277.871850
30	2.120372	0.031959	4278.868841
40	2.808837	0.038107	4279.852769
50	3.605067	0.045120	4280.892758

time: 17.5 ms (started: 2021-12-27 17:44:17 +00:00)

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf



6. Experimental Analysis of the Correctness (Functional Testing)

```
#### correctness
total = 0
lastList= []
checkA = A
while len(A)>0:  #O n
 total+=1
 cliqueList = list()
 length = len(A)
 C = list()
 maxSize = 0
 j=0
 for tempi in range(length): #O n
   i = tempi #i =1
   N1 = i #n1 = 1
   alone = True
   for singleVertex in range(length): #O n
     if A[i][singleVertex] == 1:
       alone = False
   if alone:
     C.append(i)
     cliqueList.append(C)
   for  tempj in range(length): #O n
     j = tempj
     while j<length: #O n
       if j in C:
```


1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
          j+=1
          continue
        elif i == j:
          j+=1
          continue
        elif A[i][j] == 1:
          N2 = j
          if len(C) > 0:
            for element in C: #O K
              if A[element][j] != 1:
                j+=1
                break
              elif element == C[-1]:
                C.append(j)
                i=j
                j=0
                break
          else:
            C.append(N1)
            C.append(N2)
            i = j
            j = 0
        else:
          j+=1
      if maxSize < len(C):
        maxSize = len(C)
      if len(C) > 0 and C not in cliqueList:
        cliqueList.append(C)
      C = list()
      i=tempi
  #print(maxSize," = k ")
  #O (N^3 * K)
  #print(cliqueList)
  removeCopies(cliqueList)#O n^2
  print(cliqueList)
  max_index = findMax() #O n
  lastList.append(max_index)
  quickSort(max_index,0,len(max_index)-1)#O n^2
  A = removeMax(max_index,A)#O n
print("k = ",total)
```

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

```
print(lastList)
def checkCorrection(M,subsetcheck):
 clique = True
 for element in subsetcheck:# [1]
   for other in subsetcheck:
     if element != other and M[element][other] == 0 :
       clique = False
 return clique
testCount = 10
for element in lastList:
 if not checkCorrection(checkA,element):
   print("this algorithm is wrong")
   break
 else:
   checkA = removeMax(element,checkA)#O n
print("everything is ok.")
```

```
k =  0
[]
everthing is ok.
time: 94.8 ms (started: 2021-12-27 19:31:08 +00:00)
```

We are checking all subsets which the greedy algorithm creates and see if they are considered as cliques or not. If all of the subsets are cliques, we know that our greedy algorithm works correctly.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

Ratio Bound (Black Box)

```
###ratio bound

def ratioBound(kListBruteForce, kListGreedy):
 ratioList = []
 for i in range(len(kListBruteForce)):
   ratioList.append(kListBruteForce[i] / kListGreedy[i])
 print(stat.mean(ratioList))




ratioBound(kListBruteForce, kListGreedy)
```

```
                5 vertices  6 vertices  7 vertices
#of Graphs
100                      1    0.966667     0.98050
500                      1    0.978000     0.96550
1000                     1    0.979417     0.96905
time: 19.6 ms (started: 2021-12-27 20:28:41 +00:00)
```

In ratio bound, we are comparing our two algorithms which are brute force and greedy. We are finding the ratio between their k values, the more the ratio gets closer to 1 the more greedy algorithm reaches to optimal solution.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf

7. Discussion

As a result of all the calculations, we can conclude that our algorithm works properly. In our theoretical analysis, we found that our brute force algorithm works exponential and our greedy algorithm works polynomially, and we confirmed this in our experimental analysis.

Brute force algorithm works more accurate than greedy algorithm, however it can be said that it is not as time efficient as greedy algorithm. But still, the ratio bound is very close to 1, which shows that our greedy algorithm is an optimal solution.

1. https://dl.acm.org/doi/pdf/10.5555/1379361.1379375
2. https://cpb-us-w2.wpmucdn.com/sites.udel.edu/dist/d/5653/files/2017/02/2MSC-1yg21zs.pdf