World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:9, 2008

A Deterministic Polynomial-time Algorithm for the Clique Problem and the Equality of P and NP Complexity Classes

Zohreh O. Akbari

Abstract**—**In this paper a deterministic polynomial-time algorithm is presented for the Clique problem. The case is considered as the problem of omitting the minimum number of vertices from the input graph so that none of the zeroes on the graph's adjacency matrix (except the main diagonal entries) would remain on the adjacency matrix of the resulting subgraph. The existence of a deterministic polynomial-time algorithm for the Clique problem, as an NP-complete problem will prove the equality of P and NP complexity classes.

Keywords**—**Clique problem, Deterministic Polynomial-time Algorithm, Equality of P and NP Complexity Classes.

I. Introduction

REGARDING the solving time, problems are divided in 3 categories [1]:

- **P or Tractable problems:** *Problems which are solvable by polynomial–time algorithms.*
- **Intractable problems:** *Problems proven unsolvable by polynomial–time algorithms.*
- **NP problems:** *Problems that No polynomial-time algorithm has yet been discovered for them, nor has anyone yet been able to prove that they are intractable; but the positive solution to their corresponding decision problem can be verified in polynomial time given the right information [2].*

To study the relation between the P and NP complexity classes, the concept of "NP-completeness" is very useful. This concept was first introduced by Stephen A. Cook [3]. According to [1] and [4], problem C is NP-complete if:

- *C is NP.*
- *A α C*[1] *for any A in NP.*

Therefore:

- *B α C for any B in NP-complete.*

Considering the definition of NP-complete problems, a deterministic polynomial-time solution to any of them would also be a deterministic polynomial-time solution to every other problem in NP. Thus it is sufficient to present a polynomial-time algorithm for any NP-complete problem to prove that P=NP, as mentioned in [4] and [5]. The NP-complete problem considered in this paper is the Clique problem which is one of Richard Karp's 21 problems as shown NP-complete in his 1972 paper "Reducibility among Combinatorial Problems" [6].

II. The Clique Problem

Any complete subgraph of an undirected graph is called a clique. The size of a clique is the number of vertices it contains. The Clique problem is the problem of finding a clique of maximum size in an arbitrary undirected graph. This problem is NP-complete according to [1], [4] and [6] since it is NP and:

The Boolean Satisfiability Problem α The Clique Problem[2]

In this paper a deterministic polynomial-time algorithm is presented for the Clique problem which would consequently prove the equality of the P and NP complexity classes.

III. The Deterministic Polynomial-time Algorithm for the Clique Problem

A. The Basic Idea

According to the definition of the adjacency matrix of a graph [4], it is obvious that all the entries of a complete graph's adjacency matrix are 1 except for 0's on the main diagonal. Thus the Clique problem can be considered as the problem of omitting the minimum number of vertices from the graph so that none of the zeroes on the graph's adjacency matrix (except the main diagonal entries) would remain on the adjacency matrix of the resulting subgraph.

1) The Problem of Maximum Inaccessibility Submatrix of Ones

Since the 0's on the main diagonal of the adjacency matrix are not considered to be deleted in the Clique problem, a

Zohreh O. Akbari is a post graduate student at Payame Noor University, Tehran, Iran (e-mail: z.o.akbari@ gmail.com). This study is an individual pursuit of the author out of personal interest and not a part of the academic requirement of the university.

[1] '*A*' can be reduced in polynomial time to '*C*' [4] (there exists a procedure that transforms any instance of *A* into an instance of *C* with the following characteristics:

- The transformation takes polynomial time.
- The answers are the same. That is, the answer for any instance of *A* is "yes" if and only if the answer for the corresponding instance of *C* is also "yes.")

[2] According to Cook's theorem presented in [3], the Boolean Satisfiability problem is an NP-complete problem which means any problem in NP can be reduced in polynomial time to this problem. Thus the reduction of the Boolean Satisfiablity problem to the Clique problem in polynomial time will cause the reduction of any NP problem to Clique problem in polynomial time as well.

relevant adjacency matrix can be defined while working on cliques, that is a matrix same as usual adjacency matrices with like definition except for the main diagonal entries which should be all 1's. Therefore a vertex is considered to be connected to itself, and disconnection is a concept defined within different vertices while having no edge in between. This new matrix actually presents the disconnections between vertices instead of connections and it shall be called the inaccessibility matrix.

Definition 1: *Suppose that G = (V, E) is an undirected graph, assuming that the vertices are numbered 0,1,2,...,|V|-1 in some arbitrary manner, then the Inaccessibility matrix representation of a graph G consists of a |V| × |V| matrix A = [a_{ij}], in which a_{ij} = 1 if vertex i and vertex j are connected by an edge or i = j, and otherwise a_{ij} = 0.*

The size of an inaccessibility submatrix can be consequently defined as the number of vertices it contains. Thus the Clique problem is equivalent to the problem of finding an inaccessibility submatrix of maximum size, with all entries of 1's for the input undirected graph.

2) The Problem of Minimum Nil Sweeper

Section III.A.1 described that finding a clique of maximum size in an undirected graph is equivalent to the problem of finding an inaccessibility submatrix of maximum size in it, with all entries of 1's. This is actually the action of detecting the minimum set of vertices to be deleted from the graph so that the inaccessibility matrix of the remaining subgraph does not have any zeroes on its entries. This latter problem shall hereafter be called as Minimum Nil Sweeper problem.

Each zero on the inaccessibility matrix is presenting two disconnected vertices, and to omit such zero it is enough to delete one of these vertices so that the inaccessibility matrix of the remaining graph would not include that vertex and therefore the corresponding zero. That is, the 0 at a_{ij} is omitted if either i-th or j-th vertices is omitted from the graph.

Therefore the Minimum Nil Sweeper algorithm should consider all the zeroes on the inaccessibility matrix of the input graph and delete at least one of the vertices corresponding to each zero. This will make sure that the algorithm will remove all the zeroes, but since the algorithm should present the minimum set of vertices to be deleted from the graph, there should be an appropriate method of choosing the vertex to be deleted for each zero to extract the minimum set of vertices or in other words, the Minimum Nil Sweeper.

B. The Algorithm

To present the Minimum Nil Sweeper algorithm it is necessary to define some concepts.

Definition 2: *Assuming that the zeroes on the inaccessibility matrix of an undirected graph are numbered 0,1,2,...,|N|-1 in some arbitrary manner, then for k≤N, Nil(k) is the set of k first zeroes of the inaccessibility matrix according to their arbitrary order.*

Definition 3: *Assuming that N is the number of the zeroes on the inaccessibility matrix of an undirected graph, then for k≤N the Minimum Nil Sweeper(k) or MNS(k) is the minimum set of vertices to be deleted from the graph so that the inaccessibility matrix of the remaining subgraph does not include Nil(k).*

Definition 4: *Assuming N as the number of zeroes on the inaccessibility matrix of an undirected graph, for k≤N having Nil(k) and MNS(k-1), if the k-th zero is presented by (x, y) the CNS [3](k, x) and CNS(k, y) are defined as follow:*

Supposing that G=(V, E) is the input graph which its vertices are numbered 0,1,2,...,|V|-1 in some arbitrary manner and that the |V|×|V| matrix A=[a_{ij}] is the Inaccessibility matrix representation of the graph, then a Set and a Flag can be assigned to every $V_i \in V$ (for $0 \le i \le |V|-1$), defined as follow:

$$Set(V_i) = \{V_j \in V \mid a_{ij} \in Nil(k)\} \quad (1)$$

$$Flag(V_i) = \begin{cases} 0 \text{ if } V_i \notin MNS(k-1) \\ 1 \text{ if } V_i \in MNS(k-1) \end{cases} \quad (2)$$

Regarding these values the CNS(k, x) and CNS(k, y) can be computed as follow:

$$CNS(k,x) = MNS(k-1) \cup \{x\} - \{v \in MNS(k-1) \mid \forall V_i \in Set(v)-\{x\}; Flag(V_i) = 1\} \quad (3)$$

$$CNS(k,y) = MNS(k-1) \cup \{y\} - \{v \in MNS(k-1) \mid \forall V_i \in Set(v)-\{y\}; Flag(V_i) = 1\} \quad (4)$$

Remark: *The relation between the number of vertices in MNS(k-1) and the number of vertices in CNS(k, x) and CNS(k, y) is as follow:*

$$|MNS(k-1)| \le |CNS(k,x)| \le |MNS(k-1)| + 1 \quad (5)$$

$$|MNS(k-1)| \le |CNS(k,y)| \le |MNS(k-1)| + 1 \quad (6)$$

Proof of Remark: Suppose that the k-th zero is presented by (x, y), then according to (3) the difference between MNS(k-1) and CNS(k, x) arises from:

- Vertices that are in CNS(k, x) but are not in MNS(k-1): If such vertices exist they are surely members of {x}
- Vertices that are in MNS(k-1) but are not in CNS(k, x): If such vertices exist they are surely members of the set presented in (3) as follow:

$$\alpha = \{v \in MNS(k-1) \mid \forall V_i \in Set(v)-\{x\}; Flag(V_i) = 1\} \quad (7)$$

The set {x} has one member which is the vertex x, thus it can only add one vertex to CNS(k, x) which will not happen if vertex x is already a member of MNS(k-1), therefore:

$$|CNS(k,x)| \le |MNS(k-1)| + 1 \quad (8)$$

The set 'α' presented by (7) might be empty or have just

[3] Candidate Nil Sweeper

one member, since if it has more than one vertices as its members for the purpose of contradiction, those vertices could be removed from MNS(k-1) and be replaced by vertex x and produce a set smaller than MNS(k-1), which is a contradiction[4]. The size of 'α' is 1, only when $x \notin MNS(k-1)$, since assuming that the size of the set can be 1 for $x \in MNS(k-1)$ then it means a vertex exists in MNS(k-1) that the Flag of all the members of its Set is 1 which is a contradiction[5], therefore:

$$|MNS(k-1)| \leq |CNS(k,x)| \tag{9}$$

The combination of (8) and (9) results (5). Same method can be used to show (6), thus the remark is proved.

Lemma: *Assuming that Nil(N) is the set of all zeroes on the inaccessibility matrix of an arbitrary undirected graph and having the MNS(k-1) for k≤N, then MNS(k) can be computed in polynomial time as follow:*

$$MNS(k) = \begin{cases} CNS(k,x) \text{ if } CNS(k,x) \leq CNS(k,y) \\ CNS(k,y) \text{ if } CNS(k,x) > CNS(k,y) \end{cases} \tag{10}$$

Proof of Lemma: To prove the lemma, it is needed to show that:

- The output of the lemma is the MNS(k).
- The lemma takes polynomial time to compute MNS(k).

Assuming for the purpose of contradiction that V′ is the output of the lemma and |MNS(k)|<|V′|, if the k-th zero is presented by (x, y) and none of the x or y vertices were added to MNS(k-1) in computing V′[6] then V′=MNS(k-1)[7] which results to |MNS(k)|<|MNS(k-1)|. This means MNS(k) which can delete Nil(k) and therefore Nil(k-1), contains less vertices comparing to MNS(k-1) which is in contradiction with the definition of MNS(k-1). Thus the output of lemma is MNS(k) if none of the x or y vertices were added to MNS(k-1) in computing V′. But this should also be proven if MNS(k-1) does not contain any of the x or y vertices and as a result, one of them is surely added to MNS(k-1) in computing V′. Since |MNS(k)|<|V′| (assumed for the purpose of contradiction), $V_1 \subset V'$ and $V_2 \subset V\text{-}V'$ can be defined as follow:

$$|V_1| > |V_2| \geq 0 \tag{11}$$

$$MNS(k) = V' - V_1 \cup V_2 \tag{12}$$

Considering the lemma V′ is equal to one of the CNS(k, x) or CNS(k, y) sets, thus according to the remark mentioned, it is obtained that:

$$|MNS(k-1)| \leq |V'| \leq |MNS(k-1)| + 1 \tag{13}$$

Equation (13) results to one of the followings:

$$|V'| = |MNS(k-1)| \tag{14}$$

$$|V'| = |MNS(k-1)| + 1 \tag{15}$$

The combination of (12) and (14) result to:

$$|MNS(k)| = |MNS(k-1)| - |V_1| + |V_2| \tag{16}$$

which lead to followings according to (11):

$|V_1| - |V_2| > 0$
$\Rightarrow -|V_1| + |V_2| < 0$
$\Rightarrow |MNS(k-1)| - |V_1| + |V_2| < |MNS(k-1)|$

$$\Rightarrow |MNS(k)| < |MNS(k-1)| \tag{17}$$

Equation (17) is a contradiction, since if |MNS(k)| < |MNS(k-1)| then MNS(k) which can delete Nil(k) and therefore Nil(k-1), contains less vertices comparing to MNS(k-1) which is in contradiction with the definition of MNS(k-1).

The combination of (12) and (15) result to:

$$|MNS(k)| = |MNS(k-1)| + 1 - |V_1| + |V_2| \tag{18}$$

which lead to followings according to (11):

$|V_1| - |V_2| > 0$
$\Rightarrow |V_1| - |V_2| \geq 1$
$\Rightarrow -|V_1| + |V_2| \leq -1$
$\Rightarrow |MNS(k-1)| + 1 - |V_1| + |V_2| \leq |MNS(k-1)| + 1 - 1$
$\Rightarrow |MNS(k-1)| + 1 - |V_1| + |V_2| \leq |MNS(k-1)|$

$$\Rightarrow |MNS(k)| \leq |MNS(k-1)| \tag{19}$$

Equation (19) is also a contradiction, since if |MNS(k)| = |MNS(k-1)| one of the x or y vertices are surely a member of MNS(k-1) which is not possible, and if |MNS(k)| < |MNS(k-1)| then MNS(k) which can delete Nil(k) and therefore Nil(k-1), contains less vertices comparing to MNS(k-1) which is in contradiction with the definition of MNS(k-1). Thus

[4] According to definition 4 and the lemma, for k ≤ N there must not be any vertex in MNS(k) that the Flag value of its Set members are all 1's, since this means that all its corresponding zeroes in Nil(k) can be deleted by some other vertices in MNS(k) which are in its Set and have Flag values of 1's, thus this vertex do not delete any zeroes. Therefore if there exists more than one vertices in 'α' it means that there are more than one vertices which the Flag value of their Set members are all 1's except for x, that is by adding x to MNS(k-1) those vertices could be all deleted from the MNS(k-1) and a smaller MNS(k-1) could be achieved which is a contradiction.

[5] A set smaller than MNS(k-1) can be created by deleting this vertex from MNS(k-1), since the zero entries corresponding to this vertex can be all removed by some other vertices in MNS(k-1) which are in its Set and have Flag values of 1's.

[6] This means MNS(k-1) contains one of these vertices at least.

[7] Supposing that |CNS(k, x)| ≤ |CNS (k, y)| and thus V′= CNS(k, x), since x was not added to MNS(k-1) in computing V′, x ∈ MNS(k-1). Considering (3), the set 'α' is empty since assuming for the purpose of contradiction that it has a member it means there exists a vertex in MNS(k-1) that all its corresponding zeroes in Nil(k-1) can be deleted by some other vertices in MNS(k-1) which is a contradiction. Thus CNS(k, x) = MNS(k-1). Same is for CNS(k, y) supposing that |CNS(k, x)| > |CNS (k, y)|, which means V′ = MNS(k-1).

according to the contradictions of (17) and (19) it is proven that the output of the lemma is the MNS(k).

The next step is to show that the lemma takes polynomial time to compute MNS(k). The inaccessibility matrix of a graph is diagonal symmetric due to its definition. Thus to remove all the zeroes on the inaccessibility matrix it is enough to consider only the upper triangle of the inaccessibility matrix, since while concentrating on a 0 at a_{ij} whether the i-th or the j-th vertex is omitted, the corresponding row and column of that vertex, are both omitted from the inaccessibility matrix and as a result the entry at a_{ji} is also omitted. Thus the number of zeroes considered in Minimum Nil Sweeper algorithm is $(|V|^2-|V|)/2$ at the most or $O(|V|^2)$.

Since to compute each vertices' Set it is enough to read every member of Nil(k-1) like (x, y) and add y to Set(x) and x to Set(y), and to compute vertices' Flag it is enough to read MNS(k-1) and set the Flag of its members as 1; the computation of all vertices' Set and Flag will take polynomial time:

$$T(Sets) = |Nil(k-1)| = O(|V|^2) \quad (20)$$

$$T(Flags) = |MNS(k-1)| = O(V) \quad (21)$$

According to (3) and (7), the time order needed to compute CNS(k, x) is as follow:

$$T(CNS(k,x)) = T(\alpha) \quad (22)$$

To compute 'α' all the vertices of MNS(k-1) should be considered at the most, and for each of them it is needed to check the Flag of every member of its Set. Assume that the largest Set is presented by Max_Set, therefore:

$$\begin{aligned} T(CNS(k,x)) &\le |MNS(k-1)| . |Max\_Set| \\ &\le O(|V|).O(|V|) = O(|V|^2) \end{aligned} \quad (23)$$

Same is for CNS(k, y), therefore:

$$\begin{aligned} T(CNS(k,y)) &\le |MNS(k-1)|.|Max\_Set| \\ &\le O(|V|).O(|V|) = O(|V|^2) \end{aligned} \quad (24)$$

Thus the computation of MNS(k) from MNS(k-1) will take polynomial time and the lemma is proven:

$$T(MNS(k)) = 3 \times O(|V|^2) + O(|V|) = O(|V|^2) \quad (25)$$

Theorem: *Assuming that Nil(N) is the set of all zeroes on the inaccessibility matrix of an arbitrary undirected graph, then the Minimum Nil Sweeper(N) of this graph can be computed in polynomial time.*

Proof of Theorem: To prove the theorem, it is need to present a polynomial-time algorithm which can compute the MNS(N). Suppose that N is the number of zeroes on the inaccessibility matrix of an arbitrary undirected graph, then according to the lemma, MNS(N) can be computed in polynomial time having MNS(N-1) which can also be computed in polynomial time having MNS(N-2) and so on, thus MNS(1) can be computed in polynomial time having MSN(0) which is ∅ according to its definition. Therefore MNS(N) can be computed by N times repeating the method of extracting MNS(k) from MNS(k-1) presented in the lemma for any $k \le N$, and this is actually the Minimum Nil Sweeper algorithm. The Minimum Nil Sweeper algorithm, presented below, computes the MNS(N) in polynomial time which is proven in section III.C and III.D.

The Minimum Nil Sweeper Algorithm: Suppose that G = (V, E) is the input graph of the Clique problem with N zeroes on its inaccessibility matrix, then:

- Initialize MNS(0) with ∅.
- For every $V_i \in V$ $(0 \le i \le |V|)$ assume that:

$$Set(V_i) = \phi \quad (26)$$

$$Flag(V_i) = 0 \quad (27)$$

- For every zero of Nil(N) repeat the lemma method.
- After N iterations of lemma method, every vertex with Flag value of 1 is a member of MNS(N).

C. The Proof of the Algorithm

To prove the Minimum Nil Sweeper algorithm, it is needed to show that after N iterations of lemma method, MNS(N) contains the minimum set of vertices to delete Nil(N) from the inaccessibility matrix of the input undirected graph. This may be proven by "Mathematical induction":

Initial Step: *MNS(0) is ∅ and it is obvious that ∅ is the minimum set of vertices to delete Nil(0).*

Inductive Assumption: *After the (k-1)-th iteration of lemma method, MNS(k-1) contains the minimum set of vertices to delete Nil(k-1) from the inaccessibility matrix of the input undirected graph.*

Inductive Step: *After the k-th iteration of lemma method, MNS(k) contains the minimum set of vertices to delete Nil(k) from the inaccessibility matrix of the input undirected graph.*

The k-th iteration of lemma method considers MNS(k-1) as input which contains the minimum set of vertices to delete Nil(k-1) from the inaccessibility matrix of the input undirected graph according to the inductive assumption. Considering the proof of lemma, mentioned in section III.B having the MNS(k-1) for $k \le N$, the MNS (k) can be computed using the lemma method in polynomial time. Thus the inductive step is verified and as the result the algorithm proven.

D. The Time Order of the Algorithm

Suppose that N is the number of zeroes on the inaccessibility matrix of the input graph of the Clique problem and T(N) indicates the time order for computing MNS(N). Assuming 't' as the time order for each iteration of lemma method, T(N) is computed as follow:

$$\begin{aligned} T(N) &= T(N-1) + t \\ &= T(N-2) + 2t \\ &\dots \\ &= T(1) + (N-1)t \\ &= T(0) + Nt \\ &= N\,t \end{aligned} \tag{28}$$

$$T(N) = O(|V|^2) \,.\, O(|V|^2) = O(|V|^4) \tag{29}$$

Thus MNS(N) can be computed in polynomial time and the Theorem is proven.

IV. The Faster Algorithm[8] for the Clique Problem

The Minimum Nil Sweeper algorithm presented in section III.B computes the Set and Flag for all vertices each time it uses the lemma method, which is not necessary since updating is possible.

Supposing that the Set of each vertex is defined as an array of size N, the Sets of all vertices is considered as an N×N matrix which is valued after the (k-1)-th iteration as follow:

$$Set[i][j] = \begin{cases} 0 \text{ if } (V_i, V_j) \notin Nil(k-1) \\ 1 \text{ if } (V_i, V_j) \in Nil(k-1) \end{cases} \tag{30}$$

Thus, Set[i][j]=1 means $V_i \in Set(V_j)$ and $V_j \in Set(V_i)$. Assuming that the k-th zero is presented as (x, y), Set array can be updated after the k-th iteration by making the following changes in O(1) time:

$$\begin{aligned} &Set[x][y] = 1 \\ &Set[y][x] = 1 \end{aligned} \tag{31}$$

According to the remark mentioned in III.B, on each iteration of lemma, Flag value of only two vertices might change: the one that might have been added to MNS(k-1) and the one that might have been removed from MNS(k-1). Thus to update Flag values it is enough to change the Flag value of these two vertices at the most which will take O(1) time.

The algorithm also computes the CNS(k, x) and CNS(k, y) in $O(|V|^2)$ by checking all the Set's member for every vertex of MNS(k-1). But considering the size difference between MNS(k-1) and MNS(k)[9], they can also be computed faster. To improve this time, 'Zero_Counter' can be defined as an array of length |V| as follow:

$$Zero\_Counter[i] = |\{Set[i][j] = 1 \mid Flag(V_j) = 0\}| \tag{32}$$

Thus Zero_Counter[i] contains the number of 1's on the i-th row of the Set matrix that point to a vertex with Flag value of 1. Using the definitions of Set matrix and Zero_Counter, the set 'α' presented in (7), can be computed as a vertex with following specifications:

$$Set[vertex][x] = 1 \tag{33}$$

$$Zero\_Counter[vertex] = 1 \tag{34}$$

The mentioned specifications can be checked in O(1) for each vertex and since it is needed to check MNS(k-1) vertices to compute each of the CNS(k, x) and the CNS(k, y) sets, it is obvious that their computation time order is O(|V|).

The Zero_Counter array can be updated after each iteration of lemma method in O(|V|) time, since according to the remark mentioned in III.B, on each iteration of lemma, Flag value of only two vertices might change. Thus to update Zero_Counter, if a vertex has been added to MNS(k-1) in computing MNS(k), then for all vertices of the graph, it is necessary to decrease Zero_Counter value by one if the recently added vertex belongs to their Set. Same if a vertex has been removed from MNS(k-1) in computing MNS(k), then for all vertices of the graph, it is necessary to increase Zero_Counter by one if the recently removed vertex belongs to their Set.

Therefore the time order for computation of MNS(k) from MNS(k-1) with necessary updates can be decreased to:

$$t = 2 \times O(|V|) + 2 \times O(1) = O(|V|) \tag{35}$$

Thus a faster algorithm for the Clique problem could be defined with following time order according to (28):

$$T(N) = O(|V|^2) \,.\, O(|V|) = O(|V|^3) \tag{36}$$

The Algorithm: Assumed that IM is the N×N inaccessibility matrix of the input graph; and Set, Flag and Zero_Counter are arrays with all entries of 0's, the algorithm can be described as follow:

```
MNS (int N, iMatrix IM)
{
int cns_i, cns_j, zero_i, zero_j, mns = 0;
int Set[N][N];
int Flag[N];
int Zero_Counter[N];
for (int i = 0; i < N; i ++)
  for (int j = i+1; j < N; j ++)
    if (IM[i][j] == 0)
    {
      Set[i][j] = 1;
      Set[j][i] = 1;
      Zero_Counter[i]++;
      Zero_Counter[j]++;
      cns_i = mns;
      cns_j = mns;
      zero_i = -1;
      zero_j = -1;
      if (Flag[i] == 0) cns_i ++;
```

[8] The algorithm is described as a program written in a *pseudocode* that is similar in many respects to C and Java programming languages.

[9] According to the remark and lemma presented in III.B it is obvious that: |MNS(k-1)|≤|MNS(k)|≤|MNS(k-1)|+1

```
    if (Flag[j] == 0) cns_j ++;
    for (int k = 0; k < N; k ++)
      if (Flag[k] ==1)
      {
        if (Zero_Counter[k] == 1 && Set[k][i] = 1)
        {
          cns_i --;
          zero_i = k;
          break;
        }
        if (Zero_Counter[k] == 1 && Set[k][j] = 1)
        {
          cns_j --;
          zero_j = k;
          break;
        }
      }
    if (cns_i ≤ cns_j)
    {
      if (Flag[i] != 1)
      {
        mns = cns_i;
        Flag[i] = 1;
        if (zero_i != -1) Flag[zero_i] = 0;
        for (int k = 0; k < N; k ++)
        {
          if (Set[k][i] = 1) Zero_Counter[k]--;
          if (zero_i != -1)
            if (Set[k][zero_i] = 1) Zero_Counter[k]++;
        }
      }
      else
        Zero_Counter[j]--;
    }
    else
    {
      if (Flag[j] != 1)
      {
        mns = cns_j;
        Flag[j] = 1;
        if (zero_i != -1) Flag[zero_i] = 0;
        for (int k = 0; k < N; k ++)
        {
          if (Set[k][j] = 1) Zero_Counter[k]--;
          if (zero_j != -1)
            if (Set[k][zero_j] = 1) Zero_Counter[k]++;
        }
      }
      else
        Zero_Counter[i]--;
    }
  }
for (int i = 0; i < N; i ++)
  if (Flag[i] == 0)
    print i;
}
```

V. The Equality of P and NP

Sections III and IV, both present a polynomial-time algorithm for the Clique problem. A deterministic polynomial-time algorithm for the Clique problem, as an NP-complete problem is also a deterministic polynomial-time algorithm to every other NP-complete problem according to their definition. Thus since the Minimum Nil Sweeper algorithm presented in this paper solves the Clique problem in $O(|V|^3)$ time it can be claimed that every NP-problem are solvable in $O(|V|^3)$ time. Thus the complexity classes P and NP are equal.

VI. Conclusion

This paper has introduced a deterministic polynomial-time algorithm for the problem of finding the maximum clique in an arbitrary undirected graph, known as the Clique problem. The case is considered as the problem of omitting the minimum number of vertices from an undirected graph so that none of the zeroes on the graph's adjacency matrix (except the zeroes on the main diagonal) would remain on the adjacency matrix of the resulting subgraph. The Minimum Nil Sweeper algortihm, presented in this paper, computes the maximum clique in $O(|V|^3)$ time, thus it is a polynomial-time algorithm.The existence of a deterministic polynomial-time algorithm for the Clique problem as an NP-complete problem will prove that the complexity classes P and NP are equal.

References

[1] Richard E. Neapolitan and Kumarss Naimipour, "Foundations of Algorithms using C++ Pseudocode", 3rd ed., Jones and Bartlett Publishers, 2003, ch. 9.

[2] Michael Sipser, "Introduction to the Theory of Computation", 2nd ed., International Edition, Thomson Course Technology, p 270, definition 7.19 and theorem 7.20, 2006.

[3] Stephen A. Cook, "The complexity of theorem-proving procedures", Proceedings of Third Annual ACM Symposium on Theory of Computing, pages 151–158, 1971.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, "Introduction to Algorithms", 2nd ed., MIT Press and McGraw-Hill, 2001, ch. 22 and ch. 34.

[5] Stephen A. Cook, "The P versus NP problem". Manuscript prepared for the Clay Mathematics Institute for the Millennium Prize Problems, 2000.

[6] Richard M. Karp, "Reducibility among combinatorial problems", In R. E. Miller and J. W. Thatcher (editors): Complexity of Computer Computations, pages 85–103, New York: Plenum Press, 1972.

Zohreh O. Akbari was born in Tehran, Iran, in 26 Feb 1983. She achieved her B.Sc degree in computer software engineering at technical and engineering department of Azad University, Tehran, Iran in 2006. She is a Master student from 2006 in computer software engineering at Payame Noor University, Tehran, Iran. Right now she's working on her thesis on evaluating frameworks for agent-oriented methodologies. Her research interests are software engineering methodologies, algorithms time order and expert systems.