Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 21, Number 1 (2025), pp. 51-73
©Research India Publications
http://www.ripublication.com/gjpam.htm

A PROBLEM IN MOON-MOSER GRAPHS TO SHOW P ≠NP

R. Dharmarajan[1,*] and D. Ramachandran[2]

Niels Abel Foundation, Palakkad 678011, Kerala, India.
Copyright (from 2024) R. Dharmarajan

Abstract

The P versus NP problem, a conjecture formulated by Stephen Cook in 1971, is one of the deepest and most challenging problems in contemporary mathematics and theoretical computer science. A concise mathematical formulation of the problem reads: is P = NP. In longer phrasing, this asks: given a problem instance, if some additional data can be recognized fast enough as logically implying the existence of a solution (to the instance), then can a solution be computed fast enough without the aid of any such additional data? In this article we explain why P ≠NP.

Keywords: Algorithm; polynomial time; certificate; maximal clique; Moon-Moser graph; power set. Mathematics Subject Classification 2020: 03C70, 05C69, 11Y16, 68W40

1 **Introduction**

More on terminologies, symbols and notations used in this article can be found in [3], [5] or [7]. Throughout this article, N will denote the set of positive integers, W the set of non-negative integers (i.e., W = N∪{0}), Z the set of integers and R the set of real numbers.

This article is organised as follows: Section 1 deals with some salient points on algorithms, the problem classes P and NP, proposed solutions and feasible solutions. Section 2 presents graph theoretical concepts essential for the objective of the article. Section 3 presents the main problem and Section 4 deals with an algorithm, both instrumental in proving P ≠NP. Section 5 outlines proof that this problem is in NP but not in P.

1.1 Algorithms

An algorithm is any well-defined computational procedure that takes finitely many quantities as input and produces finitely many quantities as output. An algorithm is thus a sequence of well-defined computational steps transforming the input into the output [3].

An instance (also called input instance or problem instance) of a problem is an input satisfying all the constraints in the problem statement needed to compute a solution to the problem [3]. For example, suppose the problem is to arrange a given finite sequence of positive integers in non-descending order. Then any list consisting of finitely many positive integers is an instance of this problem. One instance is: 32, 10, 18, 31, 17, 10, 17, 7, 10.

How the size (or length) of the instance is defined depends on the problem Q being studied [3]. For example, if Q is the problem seen in the preceding paragraph, then the size of the instance given there is n = 9 (since all the repetitions have to be counted in). More examples of instances and input sizes are in (A-1) through (A-4) of Appendix A.

A "step" in an algorithm is a primitive operation executed by the algorithm. In dealing with any algorithm, there is a presupposition that every primitive operation to be executed by the algorithm is unambiguously defined. Since we furnish algorithms only in lines of pseudocodes, we adopt the following point of view for the notion of number of steps [3]: we assume that each execution of the jth line of the pseudocode takes tj steps (meaning, tj primitive operations).

From a computational point of view, given a problem Q, it is natural to ask if there exists, or can be developed, an algorithm that can process any given instance of Q to a required extent in such a way that the number of steps taken by the algorithm is bounded by a fixed polynomial in the size of the given instance [2]. Such an algorithm is said to process the input instance in polynomial time (or run in polynomial time; or take polynomial time to process) and is therefore called a polynomial-time algorithm. Note that an algorithm is deemed to run in polynomial time only vis-`a-vis a specified problem Q; in other words, the phrase 'the algorithm ALG runs in polynomial time' really means that there is a specified problem Q such that ALG processes each instance of Q in polynomial time.

When we say the number of steps taken by an algorithm is bounded by a fixed polynomial in the variable n, what we mean is: there exists a polynomial f (n) that is fixed for Q (meaning, in turn, that the instance X of Q can vary but f (n) does not, so long as the problem Q does not) such that given two instances (of Q) of sizes n1 and n2, the number of steps taken by the algorithm is at most f (n1) to process the n1-sized instance and at most f (n2) for the n2-sized instance.

For the purposes of this article, we consider two types of algorithms: solve-type algorithms (that find a solution, or correctly establish the absence of any solution, to each input instance) and check-type algorithms (that do not find solutions but only check out whether any additional input that is claimed to be a solution to an instance is really so). One crucial difference between these two types is: a solve-type algorithm needs only the instance of the problem as input whereas a check-type algorithm needs the instance as well as an additional input in the form of a "proposed solution" or "attempted solution" (called so since it might be a solution or not). A solution to an instance is also called a feasible solution. A solve-type algorithm is also called an exact algorithm.

Every algorithm considered in this article is assumed, or if necessary shown to be, correct in the sense that given an input instance, the algorithm halts with an output that is correct [3].

1.2 Certificate candidates and certificates

In this article, we consider two classes of problems that are important in the context of algorithms: P and NP. Informally, the class P consists of those problems that are solvable in polynomial time [2, 3]. This means to each problem in this class there exists a polynomial-time algorithm that solves each instance of the problem.

The class NP consists of those problems that are "verifiable" in polynomial time [3]. What is meant by calling a problem "verifiable" is that if, in addition to an instance of the problem, we are somehow given a "certificate" of a solution, then we could verify, using a check-type algorithm running in polynomial time, that the certificate is indeed a solution, or logically implies (i.e., confirms) the existence of a solution, to the given instance.

It is clear that a certificate is also an input component. Also, a certificate only begins as a proposed solution, and might or might not turn out to be a feasible solution [6]. We therefore wish to distinguish between these two situations.

To this end we define a certificate candidate. A certificate candidate is a component that is input along with the problem instance under consideration, the intention (of the user) being to test whether or not the certificate candidate confirms the existence of a solution to the instance. So a certificate candidate is a proposed solution that accompanies the instance.

Let Q be the problem under consideration, X a problem instance of size n and C (X) the certificate candidate that accompanies X. The result of running an appropriate algorithm on X and C (X) will be exactly one of the following three:

1.2(i) C (X) is per se a solution to X (i.e., the proposed solution turns out to be a feasible solution). 1.2(ii) C (X) per se is not a solution but it logically implies the existence of a solution, whether or not the implied solution is subsequently made explicit (i.e., the proposed solution, though not a feasible solution by itself, confirms the existence of a feasible solution). 1.2(iii) Neither is C (X) a solution nor does it logically imply the existence of a solution.

Henceforth, if any certificate candidate C (X) obeys 1.2(i) or 1.2(ii), then we will say 'C (X) yields a solution (to X)'. And by saying 'C (X) does not yield a solution (to X)' we will mean that C (X) obeys 1.2(iii). Examples to distinguish a certificate candidate obeying 1.2(i) from one obeying 1.2(ii) are in (B-1) through (B-3) of Appendix

B.

Next, “checking out” a certificate candidate is different from “verifying” it. To verify a certificate candidate C (X) is to have a check-type algorithm establish that C (X) yields a solution to the instance X. If a certificate candidate is thus verified then it is a certificate (of a solution); else it is just a certificate candidate. If a certificate candidate becomes a certificate, then we will say either ‘the certificate candidate is verified’ or ‘the certificate is verified’. Such a verification is tantamount to saying “YES” to the question whether the instance X has a solution.

On the other hand, to check out C (X) is to have a check-type algorithm decide whether or not C (X) yields a solution to X. If the algorithm checking out C (X) decides that C (X) does not yield a solution then this decision is tantamount to saying “NO” to the question whether X can be solved with the aid of C (X). This “NO” is not a conclusive response to whether X has a solution, but only rules that the certificate candidate C (X) is not a certificate. Thus:

1.2(iv) “verifying” requires a certificate candidate that yields a solution to the given instance whereas “checking out” calls for only a certificate candidate that is not required to have any additional properties / features; and 1.2(v) an algorithm is deemed to have verified a certificate candidate C (X) (meaning, C (X) is deemed to have become a certificate) only when C (X) is shown (by the algorithm) to yield a solution to X.

So, while a certificate candidate is only a proposed solution, a certificate is a feasible solution. In other words, every certificate is a certificate candidate in the first place but not every certificate candidate becomes a certificate. Further, a verification of a certificate candidate necessarily begins as an exercise of a check-type algorithm checking out the candidate but every exercise of checking out a candidate need not culminate in verification.

Thus, certificates (feasible solutions) are special cases of certificate candidates (proposed solutions).

As noted in the penultimate paragraph of Subsection 1.1, an algorithm is said to solve an instance X of a problem Q if the algorithm produces a solution (to X) if one exists or correctly establishes the absence of a solution otherwise, in either case without the need for any certificate candidate. If an algorithm solves each instance of Q then the algorithm is said to solve Q. An algorithm that solves a problem instance X and an algorithm that only checks out a certificate candidate C (X) differ primarily in the following aspect: in the former, the input is (X, n) and the output is a conclusive response to the question of a solution to X while in the latter, the input is (X, n, C (X)) and the output is an affirmative or a negative response to the question whether C (X) yields a solution to X. Note that in each of these cases, finitely many additional components in the input are allowed.

Now rises the question whether the algorithm that checks out a given certificate candidate does so in polynomial time, for this is crucial in finding out if the concerned problem is in NP. This underlines the importance of the certificate candidate in this context. Given a problem Q, what are needed to decide that Q can be included in NP? We need: one, a check-type algorithm -call it AL (Q) -and two, to each instance X (of Q), a certificate candidate C (X) that is verified by AL (Q) in polynomial time (in n, the size of X).

1.3 Remarks on algorithms recognizing feasible solutions

An algorithm is said to recognize a feasible solution if the algorithm verifies that the input certificate candidate is a certificate. In the context of NP, such recognition needs to be done in polynomial time.

Let X be an instance of a problem Q and ALG be the algorithm designed to check if an input certificate candidate is a certificate. Suppose C1 and C2 are two distinct certificate candidates. Also suppose that both C1 and C2 are feasible solutions to X.

It is desirable that ALG checks out C1 or C2, whichever is input with X. However, owing to the design of ALG the following might ensue: 1.3(i) ALG recognizes the feasible solution C1 in polynomial time and 1.3(ii) ALG fails, or takes worse than polynomial time (perhaps, exponential or factorial time [3, 9]), to recognize the feasible solution C2.

Then C2, despite being a feasible solution, is not useful in deciding if Q is in NP. So only candidates that can be checked out by the algorithm in polynomial time should be used. To aver that Q is in NP, for each instance of Q we only need one candidate that ALG recognizes as a feasible solution in polynomial time, even if there exist other feasible solutions that ALG does not recognize at all or does not recognize in polynomial time.

1.4 The exercise of looking for certificates

If a problem Q is given, then it is not difficult to come up with an instance X of a desired size n and a certificate candidate. But the same cannot be said for a certificate, even if an algorithm is available. Then how can a certificate, if at all one exists, be obtained?

Recall the part of the informal explanation of the class NP that goes '...if, in addition to an instance of the problem, we are somehow given a "certificate" of a solution, then we could verify...' (in the second paragraph of Subsection 1.2). The operative word there is 'somehow.' This clearly indicates there are no hard and fast rules as to the source of a certificate candidate or its form or the process of obtaining it; only every certificate candidate must rest on irrefutable theory that is relevant to the problem. Indeed, given an arbitrary problem instance, it is not known how to identify a certificate [6]. So it is not known how to identify a certificate candidate that will turn out to be a certificate. A certificate candidate C (X) could be readily available (i.e., prepared beforehand by someone) or could be compiled as and when the need arises. Preparing C (X) could take any amount of time polynomial or worse. But the time to prepare C (X) will not be included in the time required for the algorithm to check out C (X). This is because whoever prepares C (X) is allowed to do any immense amount of calculation beforehand, and only the results of that calculation need be written on C (X) [9].

In any quest for finding out whether a problem is in NP, one assumption (on the quester's part) is that given an instance of the problem there is a non-vacuous class of certificate candidates corresponding to this instance.

1.5 Remarks on P and NP

The class P is contained in the class NP [2, 3]. But it is not known whether these two classes are the same -and this is the crux of the famous problem "Is P = NP?" (also called the "P versus NP" problem).

One approach to this problem is trying to prove NP is contained in P; this, if successful, gives P = NP. An opposite approach is finding or formulating a problem that falls in NP but cannot be in P (thereby establishing P ≠NP), even if such a problem is artificially formulated [8].

Any problem in NP can be solved by exhaustive search (also called perebor, meaning "brute-force search" [8]). But steps in exhaustive search grow to forbiddingly large numbers even for a moderate growth in the size of the instance. Though decades of extensive efforts to settle the P versus NP question have produced algorithms that take significantly fewer steps than exhaustive search for problems in NP, an exact polynomial-time algorithm for any of these problems is yet to be. As a consequence, it is now commonly believed that P ≠NP [10].

1.6 Atomic sub-outputs

Let S be a solution to a problem instance X. Suppose S consists of finitely many definite and distinguishable objects Y1,...,Yk (for some k ∈ N) that are to be computed in a sequence. Then each of these k objects is an atomic sub-output of S.

For example, if the problem is to compute the set of all positive integers q less than a given positive integer m such that q is a perfect square, then the set S = {1, 4, 9, 16} is a solution to the instance m = 20 of this problem. Each of the four elements of S is an atomic sub-output of S. S is the only solution to the instance m = 20. The instance m = 1 has no solution.

Now consider a problem Q such that:

1.6(i) each instance of Q has a solution and

1.6(ii) each solution to a given instance (of size n) of Q consists of matomic sub-outputs (to be computed in a sequence) for some m ∈ R with m> 1.

Any algorithm that outputs a solution to a given instance of Q has to compute all the matomic sub-outputs that the solution comprises. Then as the instance size n increases, the number of steps taken by an algorithm cannot be bounded above by any polynomial in n. Consequently, if Q is in NP then it follows immediately that P ≠ NP. To confirm that Q is in NP, we need a check-type algorithm -call it AL (Q) -such that to each instance X of Q there must be obtainable at least one certificate candidate C (X) that is verified by AL (Q), in polynomial time, to yield a solution to X. Such a C (X) can be obtained from anywhere or prepared anyhow but once it is input along with X and n then from that point AL (Q) should need to do only a polynomial amount of computations to verify C (X).

We present such a problem in Section 3. The problem involves sets of maximal cliques of Moon-Moser graphs. The necessary verifications (using an appropriate polynomial-time algorithm) are in Section 4 and Section 5.

1.7 Remarks on capabilities of algorithms

There are computational capabilities algorithms are known to possess, and literature abounds with discussions on such capabilities. And there are capabilities that supposedly cannot be possessed by any algorithm, at least as of now. For instance, to date there is no known algorithm with the capability to carry out exhaustive search in polynomial time for an arbitrary input. Discussing these capabilities is not in the scope of this article. Suffice it to say that facts / suppositions about capabilities of algorithms rest on established theories. These theories may advance and then there may come along algorithms with new capabilities. We deal with the P versus NP problem in the framework of capabilities of algorithms at present.

In the preceding discussions, only informal definitions of algorithm, P and NP have been used. Their formal definitions are in [2, 3].

2 Essential graph theory

Let $n \in W$ and $b \in N$. Let $r \in W$ and $0 \leq r \leq b - 1$. Then the equation $r = n \pmod{b}$ will mean that r is the remainder upon dividing n by b; so will the expression $n \equiv r \pmod{b}$. In particular, if $r = 0$ then n is divisible by b.

A set is a collection of definite and distinguishable objects [3, 7]. Each object in a set is an element or a member of the set. It is taken for granted that if X is a given set then there is a well-formed definition that decides

conclusively whether or not two given elements of X are distinct. Also, such a definition is made explicit when necessary. There is a unique set that contains no members, and this is the empty set, denoted by φ.

The cardinality (or, size) of a set X is the number of elements in X, and is denoted by $|X|$.

Obviously $|X|\geq 0$. If $|X|\in W$ then X is a finite set; else X is an infinite set.

2.1 Cabals and Moon-Moser graphs

An undirected simple loop-free graph G is an ordered pair G =(V, E) where V is a nonempty finite set and E is a set of subsets of V such that $|A| = 2$ for each $A \in E$. Each element of V is a vertex of G and each element of E is an edge of G. The expressions $x \in V$ and $x \in G$ will both mean that x is a vertex of G. Similarly, $\{x, y\}\in G$ and $\{x, y\}\in E$ will both mean $\{x, y\}$ is an edge of G.

Let G =(V, E) be a graph. The order of G is denoted by |G| and is defined as $|G| = |V|$. Two distinct vertices x and y of G are adjacent if $\{x, y\}\in E$. If M is a nonempty subset of V then M is a clique of G if either $|M| =1$ or the vertices of M are pairwise adjacent when $|M| > 1$. A clique M is a maximal clique of G if M is not a proper subset of any clique of G. G is complete if V is a clique of G. Obviously every graph has at least one maximal clique.

Every graph considered in this article is assumed undirected, simple and loop-free and of order at least 1.

M (G) will denote set of all the maximal cliques of G. If $M^1 \in M$ (G) and $M_2 \in M$ (G) then M_1 and M_2 are distinct if and only if M_1 and M_2 are distinct subsets of V. Also, µ (G) will denote |M (G)|, the number of maximal cliques of G.

A cabal of M (G) (or a cabal from G; or, simply, a cabal, if the graph the cabal comes from is understood from the context) is a nonempty subset S of M (G) with the following property: exists $k \in N$ such that $|A| = k$ for every $A \in S$. Such a cabal will also be called a k-cabal of M (G) (or a k-cabal from G; or, simply, a k-cabal). A k-cabal is maximal if it is not a proper subset of any k-cabal. Obviously, if k and m are two distinct positive integers, then there can be a k-cabal and an m-cabal of M (G). Also, if there is a k-cabal from G then there is a maximal k-cabal from G since M (G) is a finite set.

If S is given to be a cabal of M (G) then it will be understood that S is a k-cabal for some unique $k \in N$.

For $t \in N$, we define the set Lt (G) as Lt (G)= $\{A \in M (G): |A| = t\}$. Clearly, each nonempty Lt (G) is a cabal of M (G).

Proposition 2.1. Let G be a graph and $|G| = n$. Then:

(i) Lt (G) is nonempty for at least one $t \in\{1,...,n\}$ and Lt (G)= φ for $t>n$.

(ii) If $t, s \in N$ with $t \neq$. s then Lt (G) ∩Ls (G)= φ

(iii) For each $k \in N$ such that there is a k-cabal of M (G), there is exactly one maximal k-cabal, namely, Lk (G).

Proof. (i) and (ii) follow immediately from the definition of M (G) and Lt (G).

(iii) Suppose there were two distinct maximal k-cabals Ak and Bk for this k. Then neither of these two is a proper subset of the other. Also, each is a proper subset of Ak ∪Bk. Further, Ak ∪Bk is a k-cabal. But this immediately contradicts the maximality of Ak and Bk.

Next, the set Lk (G) is clearly a k-cabal. Lk (G) is also a maximal k-cabal since every maximal clique A (of G) such that |A| = k is a member of Lk (G).

For the remainder of this section, assume G =(V, E) and |G| = n. The following results on μ (G) are due to Moon and Moser (see [4]):

(i) μ (G) $\leq 3^{n/3}$ if $n \equiv 0(\text{mod}3)$,

(ii) μ (G) $\leq 4.3^{(n-4)/3}$ if $n \equiv 1(\text{mod}3)$ and

(iii) μ (G) $\leq 2.3^{(n-2)/3}$ if $n \equiv 2(\text{mod}3)$.

The dot (.) to the right of the inequality symbols in (ii) and (iii) denotes multiplication of real numbers. G is a Moon-Moser graph if it satisfies any of the following:

(MM0) $n \equiv 0(\text{mod}3)$ and μ (G) = $3^{n/3}$.

(MM1) $n \equiv 1(\text{mod}3)$ and μ (G) = $4.3^{(n-4)/3}$

(MM2) $n \equiv 2(\text{mod}3)$ and μ (G) = $2.3^{(n-2)/3}$

Specifically, G is an MM0 graph if it satisfies (MM0), an MM1 graph if it satisfies (MM1) and an MM2 graph if it satisfies (MM2).

A graph $G_1 = (V_1, E_1)$ is isomorphic to a graph G2 = (V_2, E_2) (written $G_1 \cong G_2$) if there exists a bijective map $f : V_1 \rightarrow V_2$ with the following property: $\{x, y\} \in E_1$ if and only if $\{ f(x) , f(y)\} \in E_2$.

$G_{MM}^{(0)}$ will denote the set of all the MM0 graphs, $G_{MM}^{(1)}$ the set of all the MM1 graphs and $G_{MM}^{(2)}$ the set of all the MM2 graphs. For each j =0, 1, 2, two members G and H of $G_{MM}^{(j)}$ are deemed distinct if and only if G is not isomorphic to H. Clearly $G_{MM}^{(i)} \cap G_{MM}^{(j)} = \varphi$ whenever i and j are distinct elements of {0, 1, 2}.

We define $G_{MM} = G_{MM}^{(0)} \cup G_{MM}^{(1)} \cup G_{MM}^{(2)}$. Then G_{MM} is the set of all the Moon-Moser graphs. For each . $n \in N - \{1\}$, there exists only one Moon-Moser graph of order n (upto isomorphism) [4]. For each $j \in \{0, 1, 2\}$, $G_{MM}^{(j)}$ is an infinite set.

Proposition 2.2. Let G =(V, E) ∈G_{MM} and |G| = n ≥ 2. Then:

(i) |M|≤ n − 1 for every clique M of G.

(ii) M (G)= L_1 (G) ∪· ··∪L_{n-1} (G) and μ (G)= |L_1 (G)| + ··· + |L_{n-1} (G)|.

Proof.

(i) If G were complete, then V would be the only maximal clique of G, from which μ (G) = 1. This would mean (ia) 1=$3^{n/3}$ if n ≡ 0(mod3), (ib) 1=4.$3^{(n-4)/3}$ if n ≡ 1(mod3) or (ic) 1=2.$3^{(n-2)/3}$ if n ≡ 2(mod3). (ia) is false since n ≥ 2. If n ≡ 1(mod3) then n ≥ 4, from which we have 1 = μ (G) ≥ 4, clearly impossible, thus ruling out (ib). If n ≡ 2(mod3) then n ≥ 2, giving 1 = μ (G) ≥ 2, again an impossibility, and so (ic) is false.

(ii) Straightforward, since Lt (G)= φ if t ≥ n and Lt (G) ∩Ls (G)= φ whenever t and s are distinct elements of {1,...,n − 1}.

Proposition 2.3. Let G and n be as in Proposition 2.2.

(i) If n ≡ 0(mod3) then |Lk (G)|≥ $\frac{3^{n/3}}{n-1}$ for some L_k (G) ⊆M (G).

(ii) If n ≡ 1(mod3) then |Lk (G)|≥ $\frac{4.3^{(n-4)/3}}{n-1}$ for some L_k (G) ⊆M (G).

(iii) If n ≡ 2(mod3) then |Lk (G)|≥ $\frac{2.3^{(n-2)/3}}{n-1}$ for some L_k (G) ⊆M (G).

Note. In (i), (ii) and (iii) above, k depends on G and k ∈{1,...,n − 1}.

Proof. (i) Suppose the conclusion were false -i.e., |Lk (G)| < $\frac{3^{n/3}}{n-1}$for each k =1,...,n − 1. Invoking Proposition 2.2(ii), we get |M (G)| = |L1 (G)| + ··· + |Ln−1 (G)| < (n − 1) $\frac{3^{n/3}}{n-1}$, resulting in μ (G) < 3n/3 that in

turn contradicts G ∈$G_{MM}^{(0)}$

A similar reasoning proves each of (ii) and (iii).

Proposition 2.4. Let N(0) = {3n : n ∈ N}, N(1) = {3n +1: n ∈ N} and

N(2) = {3n +2: n ∈ W}.

(i) $\frac{3^{n/3}}{n-1}$ cannot be bounded above by anb for any positive real constants a and b as n increases over $N^{(0)}$.

(ii) $\frac{4.3^{(n-4)/3}}{n-1}$ cannot be bounded above by anb for any positive real constants a and b as n increases over $N^{(1)}$.

(iii) $\frac{2.3^{(n-2)/3}}{n-1}$ cannot be bounded above by anb for any positive real constants a and b as n increases over $N^{(2)}$.

Proof.

(i) Suppose the conclusion were false. Then there would exist real constants c> 1 and d> 1 such that $\frac{3^{n/3}}{n-1} \leq cn^d$ for all n ∈ $N^{(0)}$. Then $3n/3 < cn^{d+1}$. This leads to $\frac{n}{3} < (d + 1) \log (cn)$, giving $\frac{n}{\log(cn)} < 3(d + 1)$

(where log denotes logarithm to base 3). But this contradicts the fact that $\frac{n}{\log(cn)}$ is unbounded above as n increases over $N^{(0)}$.

(ii) If $\frac{4.3^{(n-4)/3}}{n-1} \leq cn^d$ for real constants c > 1 and d > 1 (and for all n ∈ $N^{(1)}$) then $\frac{3^{(n-4)/3}}{n-1} < cn^d$, giving $3^{(n-4)/3} < cn^{d+1}$, leading to $\frac{n-4}{\log(cn)} < 3(d + 1)$, contrary to the fact that $\frac{n-4}{\log(cn)}$ is unbounded above as n increases over $N^{(1)}$.

(iii) If $\frac{2.3^{(n-4)/3}}{n-1} \leq cn^d$ for real constants c > 1 and d > 1 (and for all n ∈ $N^{(2)}$) then $\frac{n-2}{\log(cn)} < 3(d + 1)$ would ensue, contradicting that $\frac{n-2}{\log(cn)}$ is unbounded above as n increases over $N^{(2)}$.

2.2 Discrete intervals in N

For set union (∪) and set intersection (∩), please see [7]. Let a and b be real numbers with a < b. The interval [a, b) in R is defined as [a, b)= {x ∈ R : a ≤ x < b}.

The set N[a, b) is defined as N[a, b)= N∩ [a, b)= {q ∈ N : a ≤ q <b}, and will be called a discrete interval in N.

Let x ∈ R. The ceiling of x is denoted by $\lceil x \rceil$ and is defined to be the smallest integer q such that q ≥ x. Obviously, x ∈ Z if and only if x = $\lceil x \rceil$.

Proposition 2.5. For n ∈ N− {1}, we define the positive integer Ω_n as follows:

$\Omega_n = 3^{n/3}$ if n ≡ 0(mod3), $\Omega_n = 4.3^{(n-4)/3}$ if n ≡ 1(mod3) and $\Omega_n = 2.3^{(n-2)/3}$ if n ≡ 2(mod3).

(i) If n ≤ 4 then Ω_n = n.

(ii) If n ≥ 5 then Ω_n > n.

Proof. (i) If n = 2 then $\Omega_n = 2.3^{(n-2)/3} = 2 = n$. If n = 3 then $\Omega_n = 3^{n/3} = 3 = n$. If n = 4 then $\Omega_n = 4.3^{(n-4)/3} = 4 = n$.

(ii) Each $n \in N - \{1\}$ is of the form n =3q or n =3q +1 or n =3q + 2 for appropriate $q \in N$. So the statement (to prove) can be named P (q) instead of P (n). We use induction on q to prove P (q).

Case 1: $n \equiv 0 (\text{mod} 3)$. Write n =3q. Here $q \in N$ and $q \geq 2$.

For q = 2, n = 6. Then $\Omega_n = 3^{n/3} = 9 > n$, proving P (q) for q = 2.

Induction hypothesis: Assume P (q) is true for q = m; i.e., for n =3m, $\Omega_n > 3m$ (meaning, $3^m > 3m$). For q = m + 1, n =3m + 3. Then $\Omega_n = 3^{m+1} = 3.3^m > 3.3m$ by induction hypothesis, giving $\Omega_n > 9m = 3m + 6m > 3m + 3 = n$ since $m \geq 1$.

Case 2: $n \equiv 1 (\text{mod} 3)$. Write n =3q + 1. Here $q \in N$ and $q \geq 2$.

For q = 2, n = 7. Then $\Omega_n = 4.3^{(n-4)/3} = 12 > n$, proving P (q) for q = 2.

Induction hypothesis: Assume P (q) is true for q = m, i.e., for n =3m +1, $\Omega_n = 4.3^{m-1} > 3m + 1$.

For q = m + 1, n =3m + 4. Then $\Omega_n = 4.3^m = 3.4.3^{m-1} > 3 (3m + 1)$ by induction hypothesis, giving $\Omega_n > 9m + 3 = 3m + 4 + 6m - 1 > 3m + 4 = n$ since $m \geq 1$.

Case 3: $n \equiv 2 (\text{mod} 3)$. Write n =3q + 2. Here $q \in N$ and $q \geq 1$.

For q = 1, n = 5. Then $\Omega_n = 2.3^{(n-2)/3} = 6 > n$, proving P (q) for q = 1.

Induction hypothesis: Assume P (q) is true for q = m; i.e., for n =3m +2, $\Omega_n = 2.3^m > 3m + 2$.

For q = m + 1, n =3m + 5. Then $\Omega_n = 2.3^{m+1} = 3.2.3^m > 3 (3m + 2)$ by induction hypothesis, giving $\Omega_n > 9m + 6 = 3m + 5 + 6m + 1 > 3m + 5 = n$ since $m \geq 0$.

This completes the induction, proving (ii).

Proposition 2.6. Let Ω_n be as in Proposition 2.5. To each $n \in N - \{1\}$, there corresponds a smallest $s_n \in N$ (depending on n) such that $\frac{\Omega n}{n s_n} \leq 1$.

Proof. Take sn = $[\frac{\Omega_n}{n}]$.

Proposition 2.7. Let n ∈ N, n ≥ 5 and s_n be the smallest positive integer corresponding to n such that $\frac{\Omega_n}{ns_n} \leq 1$.

Then N[1, Ω_n + 1) equals the union

$$\mathbb{N}\left[\frac{\Omega_n}{ns_n}, \frac{\Omega_n}{n(s_n-1)}\right) \cup \mathbb{N}\left[\frac{\Omega_n}{n(s_n-1)}, \frac{\Omega_n}{n(s_n-2)}\right) \cup \ \cup \mathbb{N}\left[\frac{\Omega_n}{2n}, \frac{\Omega_n}{n}\right) \cup \mathbb{N}\left[\frac{\Omega_n}{n}, \frac{\Omega_n}{n-1}\right) \cup \mathbb{N}\left[\frac{\Omega_n}{n-1}, \Omega_n + 1\right);$$

i.e., N[1, Ωn +1) =

$$\bigcup_{j=1}^{s_n-1}\left\{\mathbb{N}\left[\frac{\Omega_n}{(j+1)n}, \frac{\Omega_n}{jn}\right)\right\} \cup \mathbb{N}\left[\frac{\Omega_n}{n}, \frac{\Omega_n}{n-1}\right) \cup \mathbb{N}\left[\frac{\Omega_n}{n-1}, \Omega_n + 1\right).$$

Proof. Let D_j denote $\mathbb{N}\left[\frac{\Omega_n}{(j+1)n}, \frac{\Omega_n}{jn}\right)$, for j =1,...,$s_n - 1$.

Also let Y1 = $\mathbb{N}\left[\frac{\Omega_n}{n}, \frac{\Omega_n}{n-1}\right)$ and Y_2 = $\mathbb{N}\left[\frac{\Omega_n}{n-1}, \Omega_n + 1\right)$.

Clearly $\frac{\Omega_n}{ns_n} < \frac{\Omega_n}{n(s_n-1)} < \frac{\Omega_n}{n(s_n-2)} < \cdots < \frac{\Omega_n}{2n} < \frac{\Omega_n}{n} < \frac{\Omega_n}{n-1} < \Omega n + 1$,

from which , $\left[\frac{\Omega_n}{ns_n}, \Omega n + 1\right) = \left(\bigcup_{j=1}^{s_n-1} D_j\right) \cup Y_1 \cup Y_2$.

So $\left[\frac{\Omega_n}{ns_n}, \Omega n + 1\right) = \left(\bigcup_{j=1}^{s_n-1} \mathbb{N} \cap D_j\right) \cup (\mathbb{N} \cap Y_1) \cup (\mathbb{N} \cap Y_2)$

This, together with N $\left[\frac{\Omega_n}{ns_n}, \Omega n + 1\right) = \mathbb{N}1, \Omega n + 1)$. leads to N[1, Ωn +1) =

$$\bigcup_{j=1}^{s_n-1}\left\{\mathbb{N}\left[\frac{\Omega_n}{(j+1)n}, \frac{\Omega_n}{jn}\right)\right\} \cup \mathbb{N}\left[\frac{\Omega_n}{n}, \frac{\Omega_n}{n-1}\right) \cup \mathbb{N}\left[\frac{\Omega_n}{n-1}, \Omega_n + 1\right).$$

Proposition 2.8. Let n ∈ N and n ≥ 5. The s_n + 1 discrete intervals D_j (j =1,...,s_n − 1), Y_1 and Y_2 seen in the proof of Proposition 2.7 are pairwise disjoint.

Proof. Let k and r be distinct elements of {1,...,s_n − 1}, with k < r. Then k +1 ≤ r and so $\left[\frac{\Omega_n}{(r+1)n}, \frac{\Omega_n}{rn}\right) \cap \left[\frac{\Omega_n}{(j+1)n}, \frac{\Omega_n}{jn}\right)$ = φ, whence $D_r \cap D_k$ = φ.

Next, $Y_1 \cap Y_2 = \varphi$ since $\frac{\Omega_n}{n} < \frac{\Omega_n}{n-1} < \Omega n + 1$.

Finally, $D_j \cap (Y_1 \cup Y_2) = \varphi$ for each $j \in \{1,...,s_n - 1\}$ since

$m \in D_j \Longrightarrow m < \frac{\Omega_n}{n}$ and $m \in Y_1 \cup Y_2 \Longrightarrow m \geq \frac{\Omega_n}{n}$.

Corollary 2.9. Let $G \in G_{MM}$ and $|G| = n \geq 5$. Let S be a cabal of M (G) and $|S| = t$. Then t is an element of exactly one of the $s_n + 1$ discrete intervals D_j ($j = 1,...,s_n - 1$), Y_1 and Y_2 (seen in the proof of Proposition 2.7) where s_n corresponds to n as in Proposition 2.6.

Proof. Consequence of Proposition 2.7, Proposition 2.8 and the obvious fact that $1 \leq |T| \leq \Omega_n$ for every cabal T from G.

2.3 Quasi-partition and anchor number

Consider the $s_n + 1$ discrete intervals D_j ($j = 1,...,s_n - 1$), Y_1 and Y_2 seen in the proof of Proposition 2.7. Their left ends are defined as follows: the left end of Dj is $\frac{\Omega_n}{(j+1)n}$ for $j = 1,...,s_n - 1$; that of Y_1 is and that of Y_2 is $\frac{\Omega_n}{n-1}$.

Let $G \in G_{MM}$ and $|G| = n \geq 5$. Let s_n correspond to n as in Proposition 2.6. The expression $N[1, \Omega n + 1) = Y_1 \cup Y_2 \cup \left(\bigcup_{j=1}^{s_n - 1} D_j\right)$ is the quasi-partition of $N[1, \Omega_n + 1)$ by these discrete intervals. (Some of these $s_n + 1$

discrete intervals may be empty; hence the term 'quasi-partition'.)

Let S be a cabal from G. The anchor number of S relative to the above quasi-partition of $N[1, \Omega_n + 1)$ is denoted by a (S) and is defined to be the left end of the unique discrete interval (among Y_1, Y_2, D_j for $j = 1,...,s_n - 1$) to which $|S|$ belongs (see Corollary 2.9).

Note. The only quasi-partition of $N[1, \Omega_n + 1)$ considered in this article is $N[1, \Omega_n + 1) = Y_1 \cup Y_2 \cup \left(\bigcup_{j=1}^{s_n - 1} D_j\right)$ where Dj ($j = 1,...,s_n - 1$), Y_1 and Y_2 are as in the proof of Proposition 2.7. Henceforth, if S is a cabal from G then a (S) will mean only the anchor number of S relative to the above quasi-partition of $N[1, \Omega_n + 1)$.

Proposition 2.10. Let $G \in G_{MM}$ and $|G| \geq 5$. Then:

(i) a (T) $\leq \frac{\Omega_n}{n-1}$ for every cabal T from G and

(ii) there exists a cabal S from G such that a (S)= $\frac{\Omega_n}{n-1}$.

Proof. (i) follows from the fact that 1 ≤ |T| ≤ Ωn for every cabal T from G.

(ii) By Proposition 2.3, there is a maximal cabal L_k (G) from G such that $|L_k$ (G)|≥ $\frac{\Omega_n}{n-1}$. Then a (L_k (G)) = $\frac{\Omega_n}{n-1}$

Corollary 2.11. Let G be as in Proposition 2.10. There exists a cabal S from G such that a (S) ≥ a (T) for every cabal T from G.

3 The problem MAXANCHOR (MM)

A variant of a problem Q is a formulation of Q that seeks a desired type of solution without altering the import of Q. Types of variants that are widely studied and used are: optimization, computation and decision.

An optimization variant of Q is a formulation of Q that asks for a solution of an optimum measure (which is either the maximum or the minimum of the concerned measure) to each instance of Q [1].

A computation variant of Q is a formulation that asks for a solution (to each instance of Q) subject to finitely many conditions.

A decision variant of Q is a formulation of Q in which each instance admits either a 'yes' or a 'no' answer. The basic ingredients [1] of a decision variant are: the set of instances, the set of proposed solutions (i.e., certificate candidates) and the predicate that decides whether a proposed solution yields a feasible solution.

In the context of the P versus NP problem, we shall be concerned with optimization and decision variants only. It is common to formulate an optimization problem Q as a decision problem to find out if Q is in NP.

The following is an optimization problem that we name MAXANCHOR (MM): If G $\in G_{MM}$ and |G| = n ≥ 5 then find a maximal cabal L_k (G) of M (G) such that a (L_k (G)) ≥ a (S) for every cabal S from G.

A decision variant of MAXANCHOR(MM)

Inputs: (i) Problem instance G $\in G_{MM}$,

(ii) instance size n = |G| ≥ 5,

(iii) r = n (mod3),

(iv) f_G : {0, 1, 2}→ R defined by:

$f_G(0) = \frac{3^{n/3}}{n-1}$, $f_G(1) = \frac{4.3^{(n-4)/3}}{n-1}$ and $f_G(2) = \frac{2.3^{(n-2)/3}}{n-1}$.

Question: Does there exist a maximal cabal L_k (G) of M (G) such that a (L_k (G)) = f_G (r), where r = n (mod3)?

Certificate candidate: C (G) = r.

Output: YES (meaning such a desired cabal L_k (G) exists) or NO (no such cabal L_k (G) exists), as appropriate.

Note. It is mandatory that the inputs (i) through (iv) and C (G) be free from any error, as also that they be logically consistent with one another.

In Section 4, we outline an algorithm (in pseudocodes) that we name GMM MAXANCHOR and prove what are required to verify that MAXANCHOR (MM) is in NP.

4 Algorithm G_{MM} MAXANCHOR

The following algorithm will be referred to as G_{MM} MAXANCHOR. The input is (G, n, r, f_G,C (G)). G, n, r, f_G and C (G), as well as the decision question and the required output, are given in the decision variant of MAXANCHOR (MM) outlined in Section 3.

Algorithm G_{MM} MAXANCHOR

```
BEGIN
1. if r =0
2. then print "G ∈$G\frac{(0)}{MM}$. Decision: YES, there exists $L_k$ (G) ⊂M (G)
            such that a ($L_k$ (G))= $\frac{3^{n/3}}{n-1}$"and STOP
3. else if r =1
4. then print "G ∈ $G\frac{(1)}{MM}$. Decision: YES, there exists $L_k$ (G) ⊂M (G)
            such that a ($L_k$ (G))= "and STOP
5. else print "G ∈G $G\frac{(2)}{MM}$. Decision: YES, there exists $L_k$ (G) ⊂M (G)
```

```
            such that a ($L_k$ (G))= ”and STOP
6. endif
7. endif
STOP
```

Note: In the above pseudo-code, excluding BEGIN and STOP, the instructions have been numbered 1 through

7. The numbered instructions will be referred to as line 1 through line 7.

Proposition 4.1. The algorithm G_{MM} MAXANCHOR is feasible (i.e., terminates in a finite number of steps) and correct.

Proof. The algorithm makes decisions based on whether r = 0, 1 or 2. Each of these checks for r clearly terminates in a finite number of steps.

The possible outputs are all accounted for in three lines of the algorithm -namely, lines 2, 4 and 5. So the algorithm returns only finitely many outputs. Printing each decision clearly terminates in a finite number of steps.

Consequently, G_{MM} MAXANCHOR is feasible. Next, we assert its correctness.

If r = 0 then by line 2 the algorithm decides YES. This output is correct by Proposition 2.3(i).

If r = 1 then by line 4 the algorithm decides YES. This output is correct by Proposition 2.3(ii).

If r = 2 then by line 5 the algorithm decides YES. This output is correct by Proposition 2.3(iii).

Proposition 4.2. Given an input (G, n, r, f_G, C (G)), G_{MM} MAXANCHOR runs in polynomial time in n.

Proof. The total number (say, T_{MM}) of steps executed by the algorithm G_{MM} MAXANCHOR is the sum of the numbers of steps for all the lines executed. Suppose that one execution of the line j requires t_j steps and that this line is executed exactly r_j times. Then $t_j r_j$ is the number of steps consumed by the line j in one execution of the algorithm.

In one execution of the algorithm, each line is executed once if at all. Hence, for j =1,..., 7, $t_j r_j$ = t_j .

We suppose each **endif** line takes constant time, independent of n.

The number of steps required for checking the value r takes from {0, 1, 2} is bounded by n^2. Likewise for the output of the decision (lines 2, 4 and 5). So the number of steps required for each of the five lines other than the **endif** lines is bounded above by n^2, whence $T_{MM} \leq 5n^2 + 2$.

Proposition 4.3. To each instance G of MAXANCHOR (MM) there is a certificate that is verified by G_{MM} MAXANCHOR in polynomial time in the size (n) of G.

Proof. C (G) = r (where r = n (mod3)) is the required certificate.

5. MAXANCHOR(MM), P (FIN), the class NP and the class P

Proposition 5.1. MAXANCHOR (MM) is in NP.

Proof. Let G be a given instance of MAXANCHOR (MM) with $|G| = n \geq 5$. The next requirements are a check-type algorithm and a certificate candidate that is verified in polynomial time by this algorithm to confirm the existence of a solution to G.

The required algorithm is G_{MM} MAXANCHOR (Section 4) and an appropriate certificate candidate is C (G) = r (Proposition 4.2 and Proposition 4.3), where r = n (mod3).

Proposition 5.2. MAXANCHOR (MM) is not in P.

Proof. Let A be a given feasible algorithm that outputs a desired maximal cabal L_k (G) of M (G) (where G is the given instance, with $|G| = n \geq 5$) as required in the optimization variant of MAXANCHOR (MM) (Section 3).

Let $|L_k (G)| = s$. The s members of L_k (G) are the atomic sub-outputs of this solution to the instance G. Name these sub-outputs $M_1,...,M_s$ in the order that A follows in computing L_k (G).

Case 1: $n \equiv 0(\text{mod}3)$. Here $s \geq \frac{3^{n/3}}{n-1}$ (see Proposition 2.3(i), Proposition 2.10 and Corollary 2.11).

For $j = 1,...,s - 1$, having taken t_j steps for only the computation of M_j, suppose A takes another t_j+1 steps to compute M_j+1; in other words, once A executes t_j steps to compute M_j then beginning with the next step A executes t_j+1 steps to compute M_j+1, allowing that any of the already-computed sub-outputs M_1 through M_j may be used anywhere in the computation of M_j+1. Obviously, then, each $t_j \geq 1$ and $t_s \geq 1$. If T (L_k) is the total number of steps taken by A to compute and output the maximal cliques M1 through 3n/3 M_s, then $T (L_k) \geq t_1 + \cdots + t_s \geq s \geq \frac{3^{n/3}}{n-1}$. By Proposition 2.4(i), A cannot run in polynomial time, and the conclusion follows.

Case 2: n ≡ 1(mod3). Here s $\geq\frac{4.3^{(n-4)/3}}{n-1}$ (see Proposition 2.3(ii), Proposition 2.10 and Corollary 2.11). Rea- soning as in the proof of Case 1 leads to T (L_k) ≥ t_1 + ⋯ + t_s ≥ s ≥ $\frac{4.3^{(n-4)/3}}{n-1}$. The conclusion follows by Proposition 2.4(ii).

Case 3: n ≡ 2(mod3). Here s ≥ $\frac{2.3^{(n-2)/3}}{n-1}$ (see Proposition 2.3(iii), Proposition 2.10 and Corollary 2.11).

Reasoning as in the proof of Case 1 leads to T (L_k) ≥ t_1 + ⋯ + t_s ≥ s ≥ $\frac{2.3^{(n-2)/3}}{n-1}$. The conclusion follows by Proposition 2.4(iii).

6. Conclusion

P ≠NP follows from Proposition 5.1 and Proposition 5.2.

There is a caveat, though. The optimization variants of the problems MAXANCHOR (MM) and P (FIN) require computations that take exponential number of steps, as can be gauged from the proof of Proposition 5.2, leading to a thought that no algorithm is likely to possess the capability to compute exponential (or worse) number of atomic sub-outputs of the solution to any instance of either problem in polynomial time. But what if underlying theories get advanced sufficiently so that algorithms with this capability are designed? Would it imply that exhaustive search could be done in polynomial time? Then would P = NP ensue? This is a moot point. However, at present, surveys ([10], for one) seem to favour the opinion that no algorithm is ever likely to have such a capability. This seems to justify the conclusion above.

Acknowledgements

This research was supported financially by Niels Abel Foundation, Palakkad, Kerala State, India.

Appendix

A. Examples of input instances and input sizes

(A-1) If the problem is to find whether a positive integer r> 1 is composite, then each r $\in \mathbb{N}-\{1\}$ is an instance. The input size of an instance r is $\lceil\log(r)\rceil$.

(A-2) If the problem is to reorder (permute) the digits of a given positive integer r ≥ 10, with the reordering subject to finitely many conditions, then each positive integer r ≥ 10 is an instance. Its size is n is the number of digits in r (with all the repetitions counted in).

(A-3) Let G =(V, E) be a graph and $S \subset V$ with $S \neq \varphi$.Then S is an independent set of G if no two elements of S are adjacent in G. Suppose |G| =2p (where $p \in N$) and the problem is to find whether G contains an independent set of cardinality p. Then each graph of even order is an instance. The input size can be the |V | or |V |+|E|.

(A-4) Suppose it is required to find whether a given finite string X of English lowercase alphabets can be reordered to obtain a meaningful English word. Then the size of an instance is the number of characters (with all the repetitions counted in) that constitute the instance. The instance tatotirni is of size n = 9.

B. Distinct certificates for the same instance

Each of (B-1) through (B-3) shows that an instance can have two or more distinct certificate candidates that become certificates. Please see [5] for the following terms in (B-2) and (B-3): subgraph and induced subgraph.

(B-1) Let G =(V, E) be a graph of order $n \geq 10$. A walk in G is a sequence $x_1,...,x_k$ of (not necessarily distinct) vertices of G such that $xj \neq x_j+1$ and $\{x_j ,x_{j+1}\} \in E$ for each j =1,...,k − 1. If the edges in a walk are distinct, then the walk is a trail. A trail that begins and ends at the same vertex is a circuit or closed trail. A trail in G that includes every edge of G is an Eulerian trail. A circuit in G that includes every edge of G is an Eulerian circuit. G is an Eulerian graph if it contains an Eulerian circuit.

If the vertices of a walk $x_1,...,x_k$ are distinct, then the walk is a path; the path in this case is also called an x_1 -x_k path. G is connected if there is an x-y path whenever x and y are distinct vertices of G.

Let $x \in V$. The degree of x in G is denoted by dx and is defined to be the number of vertices of G that are adjacent to x.

Now assume G is connected and |G| = n, with $V = \{x_1,...,x_n\}$. Consider the question of deciding if G is Eulerian. Let C_1 and C_2 be two certificate candidates with C1 being a closed trail in G and C2 being the sequence $dx_1, . . . , dx_n$. Note that each $dxj \in N$ (j =1,...,n) since G is connected [5].

Next, suppose an algorithm (say, ALG_1) has, in polynomial time, confirmed that C1 includes every edge of G. Then C_1 is an Eulerian circuit in G -i.e., C_1 is per se a solution to the problem instance G.

Next, suppose another algorithm (say, ALG_2) has, in polynomial time, confirmed that each of the n numbers dx_1 through dx_n is an even number. Then G is Eulerian ([5], pp.56). So C_2, though not an Eulerian circuit, confirms the existence of an Eulerian circuit in G. Hence C_2 yields a solution to the instance G.

Thus C_1 and C_2 are distinct (i.e., $C_1 \neq C2$) certificate candidates that become certificates for the instance G.

(B-2) Let G = (V, E) and $|G| = n \geq 10$. Suppose it is required to decide if G has an independent set of a specified cardinality $q \in N$ where $2 \leq q<n$. Consider two certificate candidates, C_3 and C_4 such that $C_3 \subset V$, $|C_3| = q$, $C_4 \subset V$ and $|C_4| = n - q$.

Next, suppose an algorithm (say, ALG_3) has, in polynomial time, verified that no two elements of C_3 are adjacent. Then C3 is an independent set of size q.

Note that the deletion of each element of C_4 from G results in the deletion of a non-negative number of edges from G.

Next, suppose another algorithm (say, ALG_4) has, in polynomial time, deleted all the elements of C_4 from G and also confirmed that in this process all the edges of G have been deleted. Then ALG_4 has, in effect, generated the induced subgraph $G - C_4$ of G and also implied that $G - C_4$ has no edges. Clearly $G - C_4$ is of order q, from which it is immediate that $V - C_4$ is an independent set of size q although C_4 need not be an independent set. So C_4 may not per se be a solution but confirms the existence of a solution, namely, $V - C_4$.

Consequently, both C_3 and C_4 are distinct certificates to confirm that the G indeed has an independent set of size q.

(B-3) Let G = (V, E) be a graph and let F be a nonempty subset of E. If $y \in V$ then y is said to be covered by F if some edge {y, z} of G is in F.

Let M denote the set of all the vertices of G that are covered by F. The following are immediate: B-3(i) If $\{x, y\} \in F$ then $x \in M$ and $y \in M$, and B-3(ii) if $z \in V - M$ then $\{z, x\} \in E - F$ for each $x \in V$ that is adjacent to z in G.

Suppose the question is to decide if a given graph G =(V, E) (of order $n \geq 10$) has a clique of a specified cardinality $q \in N$ where $2 \leq q<n$. Consider three certificate candidates W_1, W_2 and W_3 such that:

B-3(iii) $W_1 \subset V$ and $|W_1| = q$,

B-3(iv) $W_2 \subset E$ and $|W_2| = q (q - 1) /2$, and

B-3(v) $W_3 \subset V$ and $|W_3| = n - q$.

Let M denote the set of all the vertices of G that are covered by W_2.

Next, assume an algorithm (say, ALG_5) has, in polynomial time, verified that the elements of W1 are pairwise adjacent in G. Then W_1 is a clique of cardinality q and so is per se solution to the problem instance G.

Next, suppose that another algorithm (say, ALG_6) has, in polynomial time, computed M and also output that $|M| = q$. It is obvious that W_2 is not a clique, but it leads to the clique M of cardinality q, aided by ALG_6. So W_2 is per se not a solution but yields a solution (viz., M) to the instance G.

Further, suppose yet another algorithm (say, ALG_7) has, in polynomial time, deleted all the elements of W3 from G. By this deletion process, ALG_7 has generated the subgraph $G - W_3$ of G. This subgraph has vertex set $V - W_3$.

Let k be the number of edges deleted from G upon deleting all the elements of W3 from G. Suppose it turns out that $|E| - k = \frac{q(q-1)}{2}$. Then the subgraph $G - W_3$ has order q (since $|V - W_3| = q$) and has exactly $\frac{q(q-1)}{2}$ edges. So $V - W_3$ is a clique (of G) of cardinality q. So W_3 is per se not a solution but yields a solution, namely, $V - W_3$, to the instance G.

Thus W_1, W_2 and W_3 are distinct certificates to confirm that G indeed has a clique of cardinality q.

That W_2, though not a solution per se (to G), yields a solution is a consequence of the following proposition.

Proposition Appendix B.1. Let $G = (V, E)$ be a simple graph of order $n \geq 2$. Let $F \subseteq E$ and $F \neq \varphi$ Let M be the set of all the vertices of G that are covered by F and let $|F| = \frac{q(q-1)}{2}$ for some $q \in N$ with $2 \leq q \leq n$. If $2\ |M| = q$ then M is a clique of G.

Proof. Suppose x and y were distinct non-adjacent vertices of M. Then the number of edges of G that have both the endpoints in M would be less than $\frac{q(q-1)}{2}$, owing to $|M| = q$. This, in the light of $|F| = \frac{q(q-1)}{2}$, forces an edge {a, b} of G to be in F for some $a \in V - M$, a patent impossibility because no vertex in $V - M$ is covered by F . So the vertices of M are pairwise adjacent.

References

[1] Bovet, D. P. and Crescenzi, P., 1994, Introduction to the theory of complexity, Prentice Hall, London, UK.

[2] Cook, S., 2000, "The P versus NP problem," Available online: http://www.claymath.org/millennium/P vs NP/pvsNP.pdf.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C., 2009, Introduction to Algorithms, MIT Press, USA.

[4] Freeth, S. A., 1985, "Compression Methods for Graph Algorithms," Ph. D. Thesis, University of Canterbury, New Zealand, pp. 21.

[5] Harris, J. M., Hirst, J. L. and Mossinghoff, M. J., 2008, Combinatorics and Graph Theory, Springer, New York, USA. Doi: 10.1007/978-0-387-79711-3.

[6] Savage, J. E., 1998, Models of Computation, Addison Wesley, Reading, USA.
[7] Stoll, R. R., 2012, Set Theory and Logic, Courier Corporation, USA.
[8] Trakhtenbrot, B. A., 1984, "A Survey of Russian Approaches to Perebor (Brute-Force Search) Algorithms," IEEE Ann. Hist. Comput. 6(4), pp. 384-400.
[9] Wilf, H. S., 1994, Algorithms and Complexity, Internet Edition, Summer.
[10] Woeginger, G. J., 2003, "Exact algorithms for NP-hard problems: A survey," Combinatorial Optimization -Proc. 5th Intl. Workshop, Aussois, France, pp. 185–207.