International Journal of Mathematics Trends and Technology
ISSN: 2231-5373/ https://doi.org/10.14445/22315373/IJMTT-V71I9P105
Volume 71 Issue 9, 36-46, September 2025

Original Article

Cubic Monotone 1-in-3 SAT Problem is Polynomial Time Solvable

Omar Kettani

Scientific Institute, Mohammed V University, Rabat, Morocco.

Corresponding Author : kettani.o@gmail.com

Received: 24 July 2025 Revised: 29 August 2025 Accepted: 12 September 2025 Published: 30 September 2025

Abstract - *This paper presents a proof demonstrating that the Cubic Monotone 1-in-3 SAT Problem, which is a variant of the Boolean Satisfiability Problem (SAT), can be solved in polynomial time. The proof focuses on establishing that this problem is polynomial time reducible to the Maximum Independent Set Problem in a bounded treewidth graph. While further verification is required to validate the correctness of the proof, this claim represents a potential breakthrough in one of the most significant open problems in computer science and mathematics.*

Keywords - *Bounded treewidth graph, Cubic Monotone 1-in-3 SAT Problem, Independence number, Polynomial Time Algorithm, P=NP.*

1. Introduction

The P=NP conjecture stands as one of the most significant and challenging open problems in computer science, concerning the relationship between two fundamental complexity classes: P, the set of problems that can be solved in polynomial time, and NP, the set of problems for which solutions can be verified in polynomial time. The question of whether every problem whose solution can be efficiently verified can also be efficiently solved has profound implications for mathematics, cryptography, optimization, artificial intelligence, and many other fields. Since its formalization in the 1970s by Stephen Cook, Richard Karp, and Leonid Levin, the P=NP problem has remained unsolved, despite extensive research efforts to either prove or disprove the conjecture.

Proving P=NP would mean finding a polynomial-time algorithm for any NP-complete problem, such as the Boolean Satisfiability Problem (SAT), which would immediately imply that all problems in the class NP can be solved efficiently. Such a breakthrough would fundamentally transform our understanding of computational complexity and open new possibilities for solving problems previously considered intractable, including many optimization, scheduling, and decision problems. Over the years, various attempts to prove P=NP have been made, with strategies ranging from direct algorithmic approaches to combinatorial and algebraic techniques. Although no successful proof has been widely accepted, these efforts have contributed valuable insights into the nature of computational hardness and problem structure.

This paper presents a proposed proof that P=NP, which focuses on demonstrating a polynomial-time algorithm for solving an NP-complete problem: the Cubic Monotone 1-in-3 SAT Problem, a variant of SAT.

The significance of proving P=NP extends beyond theoretical interest, potentially revolutionizing fields such as cryptography, where many encryption schemes rely on the assumed hardness of NP-complete problems. Therefore, the pursuit of a proof showing that P=NP is not just an academic endeavor but also a quest with real-world impact. This paper outlines the proposed proof, its techniques, and the implications of P=NP, while recognizing the need for careful verification and peer review to confirm the validity of the argument.

Recall that a set of values for the variables of a Boolean formula F is called a truth assignment for F, and a satisfying assignment is a truth assignment that implies that F is evaluated to true. A formula that has at least one satisfying assignment is called a satisfiable formula. The satisfiability problem asks in its decision version whether a given Boolean formula is satisfiable. A literal in a Boolean formula is an occurrence of a variable or its negation. A clause is a literal or the disjunction of at least two literals. A Boolean formula is in conjunctive normal form if it is expressed as the AND of some clauses. A Boolean formula is

in 3-conjunctive normal form if each clause has exactly three distinct literals. The 1-in-3 SAT Problem is defined as follows: Given a Boolean formula F in 3-conjunctive normal form, is there a truth assignment such that each clause in F has exactly one true literal?

The exact 3-Satisfiability problem (X3SAT) asks in its decision version whether there exists a truth assignment t: {0, 1}→V (F), setting exactly one literal to 1 in each clause of F. We call such an assignment t a x-model, and we denote with X3SAT the set of all exact satisfiable 3-CNF formulas. In the search version of X3SAT, one has to decide whether F ∈ X3SAT, and in the positive problem to find an x-model of F. X3SAT restricted to expressions where every variable has exactly three occurrences, without negated variables, is called Cubic Monotone 1-in-3 SAT Problem [3,6].

The present paper is organized as follows: After the introduction, some related work is presented in the next section. The method used in this work and the main result are outlined in Section 3. Finally, the last section concludes the paper.

2. Related Work

The P=NP conjecture is a central question in computer science and has inspired numerous research efforts aimed at either proving or disproving it. While no definitive proof has yet emerged to establish that P=NP, various attempts, theoretical frameworks, and related work have helped shape our understanding of the problem and suggested possible directions for proof. This section reviews some of the key related work that has contributed to the ongoing exploration of the P=NP question, focusing on attempts to demonstrate that P indeed equals NP and the insights gained from these efforts.

Cook's Theorem and the Foundation of NP-Completeness

The modern study of the P=NP problem began with Stephen Cook's [1] landmark paper in 1971, which introduced the concept of NP-completeness and established that the Boolean Satisfiability Problem (SAT) is NP-complete (Cook, 1971). Cook's Theorem showed that if SAT can be solved in polynomial time, then every problem in NP can be solved in polynomial time, implying P=NP. This result laid the groundwork for proving P=NP by suggesting that demonstrating a polynomial-time algorithm for any NP-complete problem would be sufficient. Since then, SAT and other NP-complete problems have been the focus of many attempts to show that efficient solutions can indeed be found.

Karp's Reductions and the Expansion of NP-Complete Problems

Following Cook's work, Richard Karp (1972) expanded the theory of NP-completeness [2] by showing that many other problems across different domains are NP-complete. His paper identified 21 combinatorial problems that are NP-complete, further solidifying the idea that these problems share a common computational structure (Karp, 1972). Attempts to solve P=NP have often focused on finding polynomial-time algorithms for one or more of these classic NP-complete problems. The hope is that a breakthrough for any single problem could extend to all problems in the class, thereby proving P=NP.

Levin's Work and the Concept of Universal Search Problems

Independent of Cook, Leonid Levin [4] also contributed to the theory of NP-completeness in the early 1970s by identifying "universal" NP problems that encapsulate the difficulty of all problems in NP (Levin, 1973). Levin's work showed that if a single algorithm could solve these universal search problems in polynomial time, then all problems in NP could also be solved in polynomial time. Although Levin's framework was similar to Cook's, it offered a slightly different perspective on the complexity landscape, motivating various attempts to show that such universal problems can indeed be efficiently solved.

Practical Progress with SAT Solvers

In recent decades, significant practical advances have been made in solving specific instances of SAT, which is considered the canonical NP-complete problem. Modern SAT solvers use sophisticated techniques like Conflict-Driven Clause Learning (CDCL), heuristic-based variable selection, and preprocessing to solve large, real-world SAT instances efficiently [5] (Biere et al., 2009). Although these solvers do not solve SAT in polynomial time in the general case, the success of SAT solvers in practice has motivated research into whether similar approaches could be extended to yield a general polynomial-time algorithm for SAT and other NP-complete problems. Some researchers have speculated that there could be undiscovered structural properties in typical problem instances that make them easier to solve, potentially leading toward a proof of P=NP.

Algebraic and Combinatorial Techniques

There have also been attempts to prove P=NP using algebraic or combinatorial approaches, such as polynomial identity testing or graph-theoretic methods. Researchers have explored whether certain algebraic structures or graph properties might enable polynomial-time algorithms for NP-complete problems. For example, the study of polynomial-time solvability of specific Constraint Satisfaction Problems (CSPs) has aimed to identify broader classes of problems that can be efficiently solved.

Although no general proof has been successful, these efforts have provided valuable insights into the limitations and potential strategies for proving P=NP.

Probabilistic and Heuristic Methods

Another approach has focused on probabilistic and heuristic methods that provide approximate solutions to NP-complete problems. Techniques like randomization and simulated annealing have been used to tackle specific instances of NP problems with varying degrees of success. While these methods do not offer general polynomial-time algorithms for all cases, they suggest that the average-case complexity of some NP-complete problems may be more favorable than their worst-case complexity. This has led some researchers to investigate whether a proof of P=NP could be established under certain average-case scenarios, potentially offering a new angle on the problem.

Machine Learning and Algorithm Design

Recently, there has been interest in leveraging machine learning techniques to design algorithms that could potentially solve NP-complete problems in polynomial time. By training models on large datasets of problem instances, researchers hope to discover patterns or strategies that could generalize to efficient algorithms. While this approach is still in its infancy, it offers an innovative direction for exploring the P=NP question. The possibility that machine learning could uncover previously unknown structures in NP-complete problems has motivated further research into algorithm design driven by data and learning.

Early work focused on complete, backtracking-based algorithms.

Davis-Putnam-Logemann-Loveland (DPLL) algorithm: This classic backtracking search algorithm forms the basis for many modern SAT solvers. It systematically explores a search tree, using unit propagation and pure literal elimination to prune branches. The original work by Davis, Logemann, and Loveland (1962) laid the foundation for the field. [10]

Modern Solvers and Heuristics

Modern SAT solvers have significantly improved performance by incorporating advanced heuristics and data structures.

Conflict-Driven Clause Learning (CDCL): This is the most important development in modern SAT solvers. When a conflict (a logical contradiction) is found, the solver analyzes the conflict graph to identify the root causes. It then adds a new clause (a "learned clause") to the formula to prevent the same sequence of assignments from leading to a conflict again. This technique dramatically prunes the search space. [11]

2-Satisfiability (2-SAT): While general SAT is NP-complete, 2-SAT (where each clause has at most two literals) can be solved in linear time. This is done by modeling the problem as a directed graph and checking for strongly connected components. This area of work is an important exception to SAT's general intractability. [12]

Applications and Extensions

SAT solvers are now used to solve a wide range of real-world problems.

Maximum Satisfiability (MaxSAT): This is an optimization variant of SAT where the goal is to find a variable assignment that satisfies the maximum number of clauses. This is particularly useful for problems like scheduling and resource allocation, where finding a perfect solution may not be possible. [13]

Satisfiability Modulo Theories (SMT): SMT extends SAT by integrating domain-specific theories, such as linear arithmetic, arrays, or bit-vectors. An SMT solver combines a SAT solver's ability to handle Boolean logic with a specialized solver for the given theory, making it useful for formal verification and software analysis. [14]

Conclusion

Despite the lack of conclusive proof showing that P=NP, these related works have significantly shaped the ongoing efforts to resolve the problem. From foundational theoretical developments like Cook’s Theorem and Karp’s reductions to practical progress with SAT solvers and new approaches involving machine learning, the pursuit of proving P=NP has generated valuable insights into computational complexity. The existing research underscores the need for novel techniques and interdisciplinary methods to overcome the challenges that have so far prevented a resolution of this famous conjecture.

Recently [6], the author proposed a polynomial time algorithm for solving the Cubic Monotone 1-in-3 SAT Problem as an attempt to prove the P=NP conjecture. In the present paper, a new proof is outlined to demonstrate this conjecture.

3. Materials and Methods

First, an equivalence between the Cubic Monotone 1-in-3 SAT problem and a system of linear equations over $x \in \{0, 1\}^n$, Ax=b is established, where matrix A and vector b are defined as follows:

A_ij=1 if literal j appears in clause i, and A_ij=0 otherwise

and b is the n vector $b=(1,....,1)^t$, n times.

On the other hand, the related graph G of A is defined by:

Let F∈ Cubic Monotone 1-in-3 SAT problem, and matrix A as previously defined.

Define its associated graph G=(V, E) of F by:

V is the set of n column matrix A^j of A, for j=1...n.

$E=\{(A^i,A^j) \in VxV/ A^i . A^j \neq 0\}$

Then the following results are obtained:

Proposition 1 [6]:

Finding an x-model of F∈ Cubic Monotone 1-in-3 SAT problem (i.e., a truth assignment x setting exactly one literal to 1 in each clause of F) is equivalent to solving the system of linear equations Ax=b, over variables $x \in \{0, 1\}^n$.

Proposition 2 [6]:

Let F∈Cubic Monotone 1-in-3 SAT problem, A its associated matrix, and G its associated graph on n vertices, then the minimum degree of G is 3, and the maximum degree of G is 6.

Proposition 3 [6]:

Let F∈Cubic Monotone 1-in-3 SAT, A its associated matrix, G its associated graph on n vertices, and αG) denotes its independence number. Therefore:

i) $\alpha(G) \leq n/3$

ii) F is satisfiable if and only if $\alpha(G) = n/3$

4. Results and Discussion

Proposition 4:
Let F∈Cubic Monotone 1-in-3 SAT, A its associated matrix, and G its associated graph, such that G=(V, E) is a graph on n vertices, with minimum degree 3 and maximum degree 6. Therefore, G is a $K_{1,4}$-free graph with bounded treewidth.

Proof**:**
First, observe that each row and each column of A contains exactly three ones (because each literal is contained in three clauses and each clause contains three literals).

Suppose for contradiction that G contains an induced $K_{1,4}$ with center v and leaves {a, b, c, d}.

Column v has 1s in exactly three rows, say r_1, r_2, r_3.

For each of a, b, c, and d to be adjacent to v, each must have a 1 in at least one of $\{r_1, r_2, r_3\}$.

Since there are 4 columns {a, b, c, d} and only 3 rows $\{r_1, r_2, r_3\}$, by the pigeonhole principle, at least one row must contain 1s from at least two columns among {a, b, c, d}.

But if two columns from {a, b, c, d} have 1s in the same row, they are adjacent in G, contradicting the requirement that {a, b, c, d} form an independent set.

Therefore, G cannot contain an induced $K_{1,4}$.□

Now the following lemmas will be used to prove that G, under the given conditions, has a bounded treewidth.

More precisely, it will be proved that: If G = (V, E) has maximum degree 6, contains no induced $K_{1,4}$, and each vertex belongs to at least three triangles, then tw(G) ≤ 6.

First, observe that since G is the related graph of a given formula F ∈Cubic Monotone 1-in-3 SAT, each vertex belongs to at least three triangles (representing three clauses in F).

Recall that the treewidth of a graph equals the maximum size of a potential maximal clique minus one [9]. Therefore, to prove that tw(G) ≤ 6, it suffices to show that every potential maximal clique C in G contains at most 7 vertices.

Let C be an arbitrary potential maximal clique in G. Recall that C is a potential maximal clique if and only if G[C] is chordal and for every pair of non-adjacent vertices x, y in C, every minimal x,y-separator in G is contained in C [9].

Two cases based on whether C contains a simplicial vertex will be considered.

Case 1: C contains a simplicial vertex v.

Since v is simplicial in G[C], all neighbors of v within C form a clique. Denote this set of neighbors as N_C(v) and define T = C \ N_G[v] to be the set of vertices in C that are neither v itself nor neighbors of v.

First, inequality |N_C(v)| ≥ 3 is established. Since v must participate in at least three triangles and v is simplicial in G[C], all triangles containing v that use vertices from C must consist of v and two vertices from N_C(v). The number of such triangles equals C(|N_C(v)|, 2). For v to participate in at least three triangles, it is needed that C(|N_C(v)|, 2) ≥ 3. If |N_C(v)| = 2, then C(2,2) = 1 < 3. If |N_C(v)| = 1, then C(1,2) = 0 < 3. Therefore, |N_C(v)| ≥ 3.

Lemma 1: For any vertex t in T, at most one vertex z exists in N_C(v) such that t is not adjacent to z.

Proof of Lemma 1: Let t be any vertex in T. Since t and v are non-adjacent vertices in C, and C is a potential maximal clique, every minimal t,v-separator in G must be contained in C.

In the graph G[C], any path from t to v must pass through N_C(v), as these are the only neighbors of v in C. Therefore, N_C(v) contains a t,v-separator in G[C]. Let S be a minimal t,v-separator contained in N_C(v).

Since N_C(v) is a clique and S ⊆ N_C(v), the set S is also a clique. For S to be a minimal separator, removing any vertex from S must create a path from t to v. Since N_C(v) is a clique, if two or more vertices from N_C(v) are removed, the remaining vertices still form a connected set. Therefore, a minimal t,v-separator S contained in the clique N_C(v) must equal N_C(v) minus at most one vertex.

Since S is a t,v-separator in G[C] and G[C] is an induced subgraph of G, it is known that S separates t from v in G as well. By the potential maximal clique property, S must be contained in C (which it is, since S ⊆ N_C(v) ⊆ C). Since t is not adjacent to v, and S separates them, t must be adjacent to all vertices in S. Therefore, t is adjacent to all vertices in N_C(v) except possibly one. □

Four subcases based on the size of N_C(v), which is at least 3 and at most 6, are now analyzed.

Subcase 1.1: |N_C(v)| = 6.

Each vertex u in N_C(v) has a degree of exactly 6 in G, as it uses 1 edge to connect to v and 5 edges to connect to the other vertices in N_C(v). Therefore, no vertex in N_C(v) can be adjacent to any vertex in T. However, by the lemma, each vertex t in T needs to be adjacent to at least 5 vertices in N_C(v). This is a contradiction. Therefore, T must be empty, giving |C| = 1 + 6 + 0 = 7. Since C = {v} ∪ N_C(v) ∪ T (by definition).

Subcase 1.2: |N_C(v)| = 5.

Each vertex u in N_C(v) has degree at least 5 in G[{v} ∪ N_C(v)], using 1 edge to v and 4 edges to other vertices in N_C(v). This leaves at most 1 edge per vertex for connections to T, giving a total capacity of at most 5 edges between N_C(v) and T.

If |T| = 2 with T = $\{t_1, t_2\}$, then by the Lemma, each t_i needs at least 4 neighbors in N_C(v), requiring at least 8 edges total. But we have capacity for at most 5 edges, which is impossible. Therefore, |T| ≤ 1, giving |C| ≤ 1 + 5 + 1 = 7.

Subcase 1.3: |N_C(v)| = 4.

In this case, each vertex t in T must be adjacent to at least 3 vertices in N_C(v), as established by the separator analysis.

The degree constraints for vertices in N_C(v) are now analyzed. Each vertex u in N_C(v) is adjacent to vertex v, which uses one edge. Additionally, since N_C(v) forms a clique of size 4, each vertex in N_C(v) is adjacent to the other 3 vertices within N_C(v). Therefore, each vertex in N_C(v) uses exactly 4 edges for connections within the set {v} ∪ N_C(v). Since the maximum degree in G is 6, each vertex in N_C(v) has at most 2 edges available for connections to vertices in T.

Now the total number of edges between N_C(v) and T is counted. Since there are 4 vertices in N_C(v) and each can contribute at most 2 edges to T, the total number of edges from N_C(v) to T is at most 8.

However, if the set T contains 3 vertices, and each vertex in T requires at least 3 neighbors in N_C(v), then T would need at least 9 edges connecting to N_C(v). Since 9 exceeds 8, this is impossible.

Therefore, T can contain at most 2 vertices, which gives: |C| = |{v}| + |N_C(v)| + |T| ≤ 1 + 4 + 2 = 7.

Subcase 1.4: |N_C(v)| = 3.

Let N_C(v) = {a, b, c}. Each vertex in N_C(v) uses 3 edges within {v, a, b, c}, leaving at most 3 edges each for connections to T. Each vertex in T needs at least 2 neighbors in N_C(v) by the Lemma.

If |T| = 4 with T = $\{u_1, u_2, u_3, u_4\}$, at least 8 edges between T and N_C(v) are needed, with capacity for at most 9. Therefore, exactly 8 edges exist, with each u_i having exactly 2 neighbors in {a, b, c}. The distribution must be (3, 3, 2), meaning two vertices in N_C(v) have 3 neighbors in T and one has 2.

Without loss of generality, a and b each connect to 3 vertices in T, and c connects to 2. The unique distribution is: u_1, u_2 connect to {a, b}; u_3 connects to {a, c}; u_4 connects to {b, c}.

To avoid $K_{1,4}$ with center a in {a, v, u_1, u_2, u_3}, some vertices among $\{u_1, u_2, u_3\}$ must be adjacent. To avoid $K_{1,4}$ with center b in {b, v, u_1, u_2, u_4}, some vertices among $\{u_1, u_2, u_4\}$ must be adjacent. Both require edge u_1u_2.

For chordality of the 5-cycle u_1 - u_3 - c - u_4 - u_2 - u_1, we need a chord. Since u_1, u_2 are not adjacent to c, the only possible chord is u_3u_4.

With forced edges u_1u_2 and u_3u_4, plus required cross-edges to avoid $K_{1,4}$, the degree sequence in G[T] becomes (3, 2, 2, 1) or similar. The vertex with degree 1 in G[T], say u_4, adjacent only to u_3, has exactly 3 neighbors in C: {b, c, u_3}.

For u_4 to form 3 triangles: triangle {u_4, b, c} exists (edge bc); triangle {u_4, u_3, c} exists (edge u_3c from u_3's connection to c); the third requires edge u_3b. But u_3 connects to {a, c}, not b. Therefore, u_4 participates in only 2 triangles, a contradiction.

Let us analyze what happens when T contains more than 4 vertices. In this subcase, it was established that N_C(v) consists of exactly three vertices, which we denote as a, b, and c. From the separator analysis, it is known that each vertex in T must be adjacent to at least 2 vertices in N_C(v).

First, the number of edges available for connections between N_C(v) and T is calculated. Each vertex among a, b, and c has a degree of 6 in graph G. Within the set consisting of v, a, b, and c, each of these three vertices uses exactly 3 edges. This is because each is adjacent to v, which uses one edge, and each is adjacent to the other two vertices in N_C(v), using two more edges, since a, b, and c form a triangle. Therefore, each vertex in N_C(v) has at most 3 edges remaining for connections to vertices in T.

The total edge capacity from N_C(v) to T is therefore at most 9 edges, obtained by multiplying 3 vertices by 3 available edges each.

If T contains exactly 5 vertices, then T requires at least 10 edges connecting to N_C(v), since each of the 5 vertices needs at least 2 such connections. However, we have just established that the maximum capacity is only 9 edges. Since 10 exceeds 9, it is impossible for T to contain 5 vertices.

Similarly, if T contains exactly 6 vertices, then T would require at least 12 edges connecting to N_C(v). Since this exceeds the available capacity of 9 edges, T cannot contain 6 vertices.

More generally, for any value where T contains at least 5 vertices, the number of required edges would be at least twice the size of T, which is at least 10. Since this always exceeds the available capacity of 9 edges, T cannot contain 5 or more vertices.

This problem can also be approached by considering degree constraints more directly. Even if an attempt is made to minimize edge usage by having some vertices in T adjacent to exactly 2 vertices in N_C(v) while others are adjacent to all 3 vertices in N_C(v), it still encounters impossibilities.

The most efficient distribution of edges would require at least two vertices in N_C(v) to be adjacent to all vertices in T when the size of T is at least 5. However, if two vertices in N_C(v) were each adjacent to all vertices in T, then each of these vertices would have degree at least 9 in G. This calculation comes from 3 edges within the set containing v and N_C(v), plus at least 5 edges to vertices in T, plus 1 edge to v. Since the maximum degree in G is 6, this configuration is impossible.

Conclusion of Subcase 1.4**:**

When N_C(v) contains exactly 3 vertices, the set T can contain at most 4 vertices. The impossibility of having more than 4 vertices in T follows from simple edge counting arguments. There are not enough available edges from N_C(v) to support more than 4 vertices in T. This edge capacity argument provides a clean upper bound of 4 for the size of T. The subsequent detailed analysis in the main proof, which shows that T equals 4 leads to a triangle constraint violation, is necessary only to tighten this bound from 4 to 3. Therefore $|T| \leq 3$, giving $|C| \leq 7$.

Case 2: C contains no simplicial vertex.

Lemma 2: Every chordal graph that is not complete contains at least one simplicial vertex.

Proof: Let G be a chordal graph that is not complete. Choose non-adjacent vertices u, v with a minimal separator S of minimum size. In a chordal graph, every minimal separator is a clique. Let C_u be the component of G - S containing u. Any vertex w in C_u that sees all of S is simplicial: its neighborhood is exactly $S \cap N(w) = S$, which is a clique. Such a vertex exists by minimality of S. □

Application to the present problem: Since G[C] is chordal (C is a PMC), either G[C] has a simplicial vertex (covered in Case 1), or G[C] is complete. If G[C] is complete, then C is a clique in G. Every vertex in C is adjacent to all other |C|-1 vertices in C. Since the maximum degree is 6, we have $|C|-1 \leq 6$, giving $|C| \leq 7$.

Therefore, Case 2 is reduced to a trivial case where C is a clique in G.

Conclusion: Every potential maximal clique has size at most 7; therefore, $tw(G) \leq 6$ and G has bounded treewidth. □

Note: The complete graph K_7 shows that the bound $tw(G) \leq 6$ is tight for graphs satisfying the three constraints. It can now be verified that K_7 satisfies all three constraints of Theorem 1.

First, the maximum degree constraint is checked. In K_7, every vertex is adjacent to all other vertices in the graph. Since K_7 has 7 vertices, each vertex has exactly 6 neighbors. Therefore, every vertex has a degree of exactly 6, which satisfies the maximum degree constraint.

Second, the fact that K_7 contains no induced $K_{1,4}$ can be verified. To form an induced $K_{1,4}$, it is needed 5 vertices where one vertex is adjacent to the other four, and those four vertices are pairwise non-adjacent. Consider any 5 vertices in K_7. Since K_7 is complete, these 5 vertices induce a complete subgraph K_5. In K_5, every vertex is adjacent to every other vertex, giving a total of 10 edges. In contrast, $K_{1,4}$ has exactly 4 edges (from the center to each of the 4 other vertices). Since K_5 and $K_{1,4}$ have different edge counts and different structures, K_5 cannot be isomorphic to $K_{1,4}$. Therefore, K_7 contains no induced $K_{1,4}$.

Third, the fact that each vertex belongs to at least three triangles can be confirmed. Consider any vertex v in K_7. This vertex has 6 neighbors. Since K_7 is complete, any two neighbors of v are connected by an edge. Therefore, for each pair of neighbors of v, a triangle containing v and that pair exists. The number of such pairs equals C(6,2) = 15. Thus, each vertex v belongs to exactly 15 triangles. Since 15 is greater than 3, the triangle constraint is satisfied.

Having verified all three constraints, it can be concluded that K_7 is a valid example of a graph satisfying the conditions of the Theorem.

Since K_7 satisfies all three constraints and has treewidth exactly 6, no bound smaller than 6 would be valid. The Theorem states $tw(G) \leq 6$, and K_7 shows this bound is achieved, making it optimal.

Note that the Maximum Independent Set (MIS) problem is NP-hard for $K_{1,4}$–free graphs, but becomes polynomial-time solvable in bounded treewidth graphs.

Theorem: The MIS problem is polynomial-time solvable for a bounded treewidth graph.
Proof: Consider this improved version of Bodlaender's algorithm [7], which computes an MIS in a bounded treewidth graph:

Table 1: pseudo-code of algorithm 1 for solving the MIS problem in a bounded treewidth graph

Algorithm: ComputeMIS(G)

Input: Graph G = (V,E)

Output: Size of the maximum independent set in G

1. (T, {Xt}t∈V(T)) ← ComputeTreeDecomposition(G) // T is a tree, Xt is the bag for node t ∈ V(T) // Root T, arbitrarily at node r
2. For each t ∈ V(T) and S ⊆ Xt: Define dp[t,S] to store the maximum size of an independent set I where:
    - I ∩ Xt = S
    - I ⊆ V(Gt) where Gt is the subgraph induced by the vertices in t's subtree

3. For each $t \in V(T)$ in post-order traversal: For each $S \subseteq Xt$:

   If S is not independent: $dp[t,S] = -\infty$ Else: If t is a leaf: $dp[t,S] = |S|$ Else: $dp[t,S] = |S| + \sum\{\text{child c of t}\}$ best(t,c,S) where $best(t,c,S) = \max\{dp[c,S'] - |S \cap S'| : S' \subseteq Xc, S' \text{ is independent}, S \cap Xc = S' \cap Xt\}$

4. Return $\max\{dp[r,S] : S \subseteq Xr\}$

Corollary of Theorem: The algorithm ComputeMIS(G) correctly computes the size of the maximum independent set in G.

Proof: The correctness of this dynamic programming algorithm will be proved by induction on the height of the subtree rooted at node t in the tree decomposition.

Base Case: Consider when t is a leaf node in the tree decomposition. For any independent set $S \subseteq Xt$, we set $dp[t,S] = |S|$. This assignment is correct because the subgraph Gt at a leaf node consists only of vertices in Xt, so Gt = G[Xt]. Any non-independent set S will be assigned $dp[t,S] = -\infty$, correctly indicating that such sets cannot be part of a valid solution.

Inductive Hypothesis: For all nodes at height less than h, assume that dp[t,S] correctly computes the maximum independent set I size in the subgraph Gt where $I \cap Xt = S$.

Inductive Step: Consider a node t at height h with children $c_1,...,c_k$, and let S be an independent set in Xt. It will be proved that dp[t,S] correctly computes the maximum independent set size in two parts: achievability and optimality.

Part 1 (Achievability): First, it will be proved that dp[t,S] represents an achievable independent set size. For each child ci of t, let Si' be the subset of Xci that achieves best(t, ci, S), and let Ii be the independent set in Gci that achieves dp[ci,Si']. An independent set $I = S \cup (\cup i\ Ii \backslash Xci)$ will be constructed, and two properties will be proved:

1) I is independent in Gt because:

- S is independent of the selection criteria

- Each Ii is independent of the inductive hypothesis

- The tree decomposition properties ensure there are no edges between vertices in different children's subtrees

- The consistency condition $S \cap Xc = S' \cap Xt$ ensures no edges exist between S and Ii\Xci

2) The size calculation $|I| = dp[t,S]$ is correct because $|I| = |S| + \sum i\ (|Ii| - |S \cap Si'|)$, which exactly matches our dynamic programming formulation.

Part 2 (Optimality): To prove that dp[t,S] is optimal, consider any independent set J in Gt where $J \cap Xt = S$. For each child ci, define $Ji = J \cap V(Gci)$ and $Si' = Ji \cap Xci$. By the inductive hypothesis, it is known that $dp[ci,Si'] \geq |Ji|$. Therefore, the dynamic programming value dp[t,S] must be at least |J|, proving optimality.

The algorithm maintains three crucial properties:

1) The tree decomposition properties ensure that all edges in the graph are considered

2) The post-order traversal guarantees children are processed before their parents

3) The consistency check $S \cap Xc = S' \cap Xt$ maintains valid solutions across adjacent bags

At the root node r, the algorithm correctly computes the maximum independent set size because:

1) Every independent set I in G defines some subset S = I ∩ Xr

2) The tree decomposition covers all vertices in G

3) The dynamic programming table considers all valid combinations

The algorithm achieves a time complexity of O(n · 2^(2k+2)), where n is the number of nodes in the tree decomposition and k is its width. This complexity arises from processing O(2^k) subsets for each bag and spending O(2^k) time combining solutions from children. The space complexity is O(n · 2^k), representing the storage needed for our dynamic programming table.

Therefore, it was proved that the algorithm correctly computes the maximum independent set size in polynomial time for graphs of bounded treewidth.

Corollary 1 of Proposition 4: There exists a polynomial time algorithm that decides whether a formula F∈Cubic Monotone 1-in-3 SAT problem is satisfiable.

Proof:

Consider the following algorithm:

Table 2: pseudo-code of proposed algorithm for solving the Cubic Monotone 1-in-3 SAT problem

INPUT: F∈Cubic Monotone 1-in-3 SAT problem
Step 1: Define the A matrix of F.
Step 2:Calculate det(A), if det(A) ≠0 then OUTPUT(' F has a unique solution: $x=A^{-1}b$ ')
else goto step 3.
Step 3: Construct the associated graph G of F.
Step 4: Apply MIS algorithm 1 to G.
if α(G)=n/3 then OUTPUT(' F is satisfiable ') else OUTPUT(' F is not satisfiable ')

Where α(G) denotes the independence number of G, and MIS algorithm 1 is a polynomial-time algorithm which computes a maximum independent set in bounded treewidth Graphs (see Table 1).

The correctness of this algorithm results from the previous propositions. Indeed, it starts by computing det(A); if it is≠0, then the algorithm outputs the unique solution of the problem: the vector $x=A^{-1}b$ (proposition 1), else it constructs the associated G graph of the input formula. Then, since G is a bounded treewidth graph, it can apply MIS algorithm 1 to G. Hence, if α(G)=n/3 then it outputs that F is satisfiable, else it outputs that F is not satisfiable by proposition 3.

Observe that it can be found x satisfying F (a truth assignment x setting exactly one literal to 1 in each clause of F: x is defined by x_i=1 if vertex i ∈ is in the maximum independent set in G and x_i=0 otherwise).

Note that Steps 1 and 3 can be done in $O(n^2)$ time. Step 2 requires $O(n^3)$ time. Meanwhile, Step 4 requires O(n) time (according to Proposition 4). Thus, the overall complexity of this algorithm is $O(n^3)$.

Therefore, the Cubic Monotone 1-in-3 SAT problem is in P.

Corollary 2 of Proposition 4: P=NP.

Since the Cubic Monotone 1-in-3 SAT problem is in P and this problem is NP-complete [3], then the complexity classes P and NP are equal [1,8] and the P=NP conjecture holds.

5. Conclusion

In this paper, a polynomial time algorithm that decides whether a formula F∈Cubic Monotone 1-in-3 SAT problem is satisfiable was proposed. Since this problem is NP-complete, it follows that the P=NP conjecture is true. The proposed proof that P=NP represents a potential breakthrough in one of the most profound and longstanding questions in computer science. By presenting a polynomial-time algorithm for an NP-complete problem such as the Cubic Monotone 1-in-3 SAT problem, the proof aims to establish that all problems in NP can indeed be solved efficiently. The approach taken combines both algebraic and graph techniques to reduce the problem's complexity to a polynomial bound, challenging the traditional understanding of computational intractability. If validated, this result would fundamentally change the landscape of computational complexity, offering new solutions to problems across various domains, from optimization and scheduling to artificial intelligence and cryptography.

Acknowledgments

The authors would like to express their gratitude to the anonymous referee for their valuable time in reviewing the manuscript.

References

[1] Stephen A. Cook, *The Complexity of Theorem-Proving Procedures*, pp. 1-8, 1971. [Google Scholar] [Publisher Link]

[2] Richard M. Karp, *Reducibility among Combinatorial Problems*, University of California, 1972. [Google Scholar] [Publisher Link]

[3] C. Moore, and J.M. Robson, "Hard Tiling Problems with Simple Tiles," *Discrete & Computational Geometry*, vol. 26, pp. 573-590, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[4] A. Leonid Levin, "Universal Search Problems," *Problems of Information Transmission*, vol. 9, no. 3, pp. 265-266, 1973. [Google Scholar]

[5] Armin Biere, Hans van Maaren, and Toby Walsh, *Handbook of Satisfiability*, IOS Press, pp. 1-980, 2009. [Google Scholar] [Publisher Link]

[6] Omar Kettani, "Solving the Cubic Monotone 1-in-3 SAT Problem in Polynomial Time," *Global Journal of Computer Science and Technology*, vol. 23, no. A1, pp. 1-10, 2024. [Publisher Link]

[7] Hans Leo Bodlaender, *A Tourist Guide through Treewidth*, Department of Computer Science, Utrecht University, pp. 1-24, 1992. [Google Scholar] [Publisher Link]

[8] Thomas H. Cormen et al., *Introduction to Algorithms*, 3rd ed., MIT Press, pp. 1-1292, 2009. [Google Scholar] [Publisher Link]

[9] Vincent Bouchitté, and Ioan Todinca, "Treewidth and Minimum Fill-in: Grouping the Minimal Separators," *SIAM Journal on Computing*, vol. 31, no. 1, pp. 212-232, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[10] Martin Davis, George Logemann, and Donald Loveland, "A Machine Program for Theorem-Proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394-397, 1962. [CrossRef] [Google Scholar] [Publisher Link]

[11] J.P. Marques-Silva, and K.A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506-521, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[12] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan, "A Linear-time Algorithm for Testing the truth of Certain Quantified Boolean Formulas," *Information Processing Letters*, vol. 8, no. 3, pp. 121-123, 1979. [CrossRef] [Google Scholar] [Publisher Link]

[13] R. Dechter, *Bucket Elimination: A Unifying Framework for Probabilistic Inference*, Learning in Graphical Models, Springer, pp. 75-104, 1998. [CrossRef] [Google Scholar] [Publisher Link]

[14] Clark Barrett, and Cesare Tinelli, *Satisfiability Modulo Theories*, Handbook of Model Checking, pp. 305-343, 2018. [CrossRef] [Google Scholar] [Publisher Link]