Essay

Polynomial Time Algorithm for Solving Sudoku Problems

Youqiang Yu *

Posted Date: 31 October 2025

doi: 10.20944/preprints202507.0434.v3

Keywords: Sudoku puzzle; NP-complete problem; polynomial-time algorithm

 doi:10.20944/preprints202507.0434.v3

Essay

Polynomial Time Algorithm for Sudoku Solving

Yu Youqiang

Institution: Hongshilazi Town Central Health Center, Jilin City Huadian City, China; 2621061585@qq.com

Abstract

The NP=P? (NP equals P) problem, as one of the seven Millennium Mathematics Problems, continues to challenge researchers in mathematics and computer science. The core of this question lies in exploring whether the complexity classes P and NP are equivalent. NP-complete problems, as the most challenging subset within NP, would prove P=NP if any NP-complete problem could be solved in polynomial time. Sudoku solving, a typical NP-complete problem, has yet to find a polynomial-time solution algorithm. This paper proposes a polynomial-time algorithm for solving Sudoku, aiming to demonstrate that NP-complete problems can be solved in polynomial time, thereby proving P=NP. The algorithm not only applies to standard 9x9 Sudoku grids but can also be extended to Sudoku grids of arbitrary size, significantly reducing computational complexity compared to non-polynomial-time algorithms. This research holds important theoretical significance for exploring the NP=P? problem.

Keywords: Sudoku puzzle; NP-complete problem; polynomial-time algorithm

1. Introduction

In computer science, the NP=P? question serves as a guiding beacon for theoretical exploration[1], directing researchers to delve into the essence of computational efficiency. This fundamental inquiry not only defines the boundaries of algorithmic capabilities but also touches upon the core paradigms of computer science. P-class problems, characterized by polynomial-time solvability, demonstrate computational efficiency, while NP-class problems, though solvable in polynomial time, may lack efficient solution processes, forming the core challenge in algorithm research. NP-complete problems[2], representing the most computationally complex subset within NP, push this challenge to its extreme. The existence of a polynomial-time algorithm for NP-complete problems would validate P=NP, a breakthrough that would revolutionize computer science. Numerous complex problems could then be efficiently resolved in polynomial time, significantly advancing global technological and economic development.

The Sudoku puzzle originated from the Latin square problem studied by 18th-century mathematician Leonhard Euler, and has been proven to be an NP-complete problem[3]. The most common variant today is the 9×9 grid Sudoku, which requires players to logically fill in numbers 1-9 to ensure uniqueness across all rows, columns, and sub-grids. Despite its simple rules, solving Sudoku is highly challenging. For puzzles with unique solutions, the complexity typically increases significantly when 17 numbers are already filled[4]. This paper proposes a polynomial-time algorithm for solving Sudoku, aiming to demonstrate P=NP by addressing this NP-complete problem. The algorithm is compatible with Sudoku puzzles of any size and promises a breakthrough in solving efficiency.

2. Current Work Progress

As a quintessential constraint satisfaction problem and a core NP-complete instance, Sudoku solving has maintained a deep connection with computational complexity theory development. By 2025, academia has established a multidimensional research framework encompassing traditional exact algorithms, heuristic optimization, mathematical modeling, and AI-driven approaches, yet the

 doi:10.20944/preprints202507.0434.v3

theoretical barrier to polynomial-time solving remains unbroken. This review systematically examines existing work through three dimensions: Sudoku complexity classification, algorithmic evolution, and specialized problem studies.

2.1. Computational Complexity of Sudoku Problem

The NP-completeness of Sudoku serves as the theoretical cornerstone of algorithmic research. In 2002, researchers first demonstrated Sudoku's NP-completeness through a reduction method—transforming the classic NP-complete problem "Latin Square Completion" into a (N×N) scale Sudoku problem 56, with the transformation process being polynomial-time computable. This proof conclusively establishes computational equivalence between Sudoku solving of any scale and Latin Square Completion, whose NP-completeness has been rigorously proven. This conclusion laid the foundation for Sudoku's complexity: if no polynomial-time algorithm exists for solving Sudoku, then P=NP; conversely, discovering such an algorithm would directly advance the proof of P=NP. Subsequent research further refined the complexity boundary: for standard 9×9 grid Sudoku, even with only 17 theoretical minimum clues (the minimum number of clues for a unique solution), the solving process still requires handling an exponential candidate solution space. For (N×N) extended Sudoku, constraint conditions grow quadratically with scale, causing exponential increases in solution space complexity, while existing algorithms exhibit super-polynomial growth characteristics in execution time.

2.2. Mainstream Paths of Sudoku Solving Algorithm

Current Sudoku solving algorithms can be divided into four categories: exact algorithms, heuristic algorithms, mathematical modeling and AI-driven algorithms. These methods complement each other in terms of solving accuracy and efficiency, but none of them can break through the inherent complexity limit of NP-complete problems.

2.2.1. Exact Algorithm: Ensures the Integrity of the Solution but Is Limited in Efficiency

The exact algorithm is the core of exhaustive search, and the constraint pruning is the core tool to verify the uniqueness of Sudoku solution.

Backtracking Method and Its Optimization[7]: As one of the most classic exact algorithms, backtracking employs depth-first search to attempt number placement. When encountering conflicts with row, column, or box constraints, it backtracks to the previous state. The basic backtracking method requires traversing numerous invalid paths when solving expert-level Sudoku puzzles (with 17-20 clues), potentially taking minutes to complete. By integrating the "Minimum Remaining Numbers Method" to prioritize filling cells with the fewest remaining candidates[8], the search space can be reduced by approximately 20%, cutting expert-level puzzle-solving time to seconds. MATLAB's backtracking solver has achieved stable performance for 9×9 Sudoku puzzles, but still exhibits significant performance bottlenecks when handling 16×16 or larger grid sizes.

Dance Chain Algorithm[9]: Designed for the exact cover problem in Sudoku, this algorithm efficiently represents constraints through sparse matrices. While outperforming basic backtracking methods in 9×9 grid puzzles, its exponential time complexity grows exponentially with matrix dimensions, making it unsuitable for solving large-scale Sudoku problems.

2.2.2. Heuristic Algorithm[10]: A Practical Solution to Balance Efficiency and Accuracy

The heuristic algorithm guides the search process by domain knowledge and sacrifices the integrity of the solution to ensure efficiency improvement, which is widely used in engineering practice.

Local Search and Evolutionary Algorithms: Genetic algorithms encode Sudoku solutions as "individuals" and optimize populations through selection, crossover, and mutation operations. By incorporating constraint checking to filter valid solutions, they achieve over 95% accuracy in solving

 doi:10.20944/preprints202507.0434.v3

moderately difficult Sudoku puzzles. However, they tend to get trapped in local optima for highly constrained puzzles. The steady-state genetic algorithm enhances convergence speed by retaining high-quality individuals, reducing solving time by 30% compared to standard genetic algorithms. Hopfield neural networks encode Sudoku constraints through energy function design, gradually converging network states to valid solutions. However, their performance heavily depends on initial parameter settings and demonstrates lower stability than evolutionary algorithms.

Advanced logic strategy integration: Formalizing human problem-solving strategies into algorithmic rules, such as the "X-Wing" and "Swordfish" chain reasoning methods. By analyzing numerical dependencies to eliminate invalid candidates, this approach can reduce the search space of complex puzzles by over 50%. The hybrid algorithm combining these methods with backtracking has become the core technology of commercial Sudoku solvers.

2.2.3. Mathematical Modeling and AI-Driven Innovation

In recent years, cross-field methods have provided a new perspective for solving Sudoku, but have not yet broken the polynomial time limit.

Linear programming modeling[11]: The Sudoku puzzle is transformed into an integer programming problem by introducing three-dimensional binary variables (v, r, c) to indicate whether cell (r, c) contains the digit v. The row, column, and box constraints are converted into 1089 linear equations. Using tools like Coin-OR PuLP, the 9×9 grid Sudoku can be solved within milliseconds. However, for a 16×16 grid, the solution time grows exponentially with variable scale, highlighting the inherent limitations of combinatorial optimization methods.

Deep learning approaches[12]: Convolutional neural networks (CNNs) can rapidly predict values for blank cells in medium-difficulty Sudoku puzzles by learning spatial features of the grid, outperforming traditional heuristic algorithms in accuracy. However, their precision drops significantly when solving extremely challenging puzzles with fewer than 20 clues. Generative adversarial networks (GANs) have made breakthroughs in Sudoku generation, automatically producing puzzles of varying difficulty levels. However, their generative logic cannot be directly applied to solving puzzles.

2.3. Polynomial Time Generation Algorithm for Sudoku Endgame

Polynomial-Time Generation Algorithms: Current research has demonstrated that Sudoku endgame configurations without numerical constraints can be generated through polynomial-time algorithms[3]. These methods typically involve constructing initial rows using the "displacement sorting method," then expanding them into complete endgames through row/column transformations, or employing generator networks to batch-produce valid configurations. While such approaches enable the generation of vast numbers of equivalent endgames through global numerical transformations, their fundamental mechanism differs from constrained solving problems. The former operates without restrictive conditions, whereas the latter requires strict adherence to specified numerical constraints.

2.4. Limitations of Existing Research

Current research on Sudoku solving faces three fundamental limitations. First, the complexity bottleneck: all exact algorithms exhibit super-polynomial time complexity as problem size increases, while heuristic methods, though efficient, cannot guarantee complete solutions. Second, poor scalability: existing algorithms are primarily designed for the 9×9 grid standard, resulting in a sharp decline in performance for 16×16 and larger grids. Third, weak theoretical connections: most studies focus on engineering implementations rather than addressing core NP-complete theory issues, lacking direct support for P=NP proofs.

 doi:10.20944/preprints202507.0434.v3

The YYQ algorithm proposed in this paper addresses the aforementioned limitations by constructing auxiliary Sudoku to establish constraint mapping relationships, aiming to overcome the theoretical barriers of polynomial-time solution.

3. Introduction of the Problem and Several Definitions

Research has confirmed that Sudoku puzzle-solving with given numbers is classified as an NP-complete problem. However, when no numerical constraints are provided, methods like the misplaced sorting algorithm can construct valid Sudoku final configurations from scratch. Building on this foundation, implementing global permutation transformations can generate multiple valid Sudoku solutions. It is evident that algorithms generating Sudoku final configurations without numerical constraints belong to polynomial-time algorithms. Taking this breakthrough as the foundation, this paper proposes a polynomial-time algorithm for solving Sudoku puzzles.

Sudoku puzzles can be categorized into three types: those with a unique solution, multiple solutions, or no solution at all. Since problems with multiple solutions or no solution can be reduced to unique-solution Sudoku problems in polynomial time, this paper focuses on polynomial-time algorithms for uniquely solvable Sudoku puzzles to prove P=NP. We introduce the YYQ algorithm, which efficiently solves uniquely solvable Sudoku problems within polynomial time. The discussion begins with defining the Sudoku solving problem.

Definition 1: For a given N-sudoku grid, the symbol N represents a natural number.

Definition 2: In a given N-grid Sudoku, the numerical operations follow an N-ary system, cycling through the range of 1 to N.

Definition 3: In a given N-sudoku grid, identical numbers are classified as a single number type.

Definition 4 For a given N-grid Sudoku, according to the Sudoku rules, the correct number in the empty cell can be deduced without further calculation based on the existing clue numbers, which is also called the clue number.

Definition 5: For a given N-grid Sudoku, the final board constructed at corresponding positions is termed the precursor auxiliary Sudoku. The digits in the unsolved Sudoku's final board and those in the precursor auxiliary Sudoku form a one-to-one correspondence.

Definition 6: For a given N-cell Sudoku, construct a final board called an auxiliary Sudoku that satisfies the following conditions:

(1) At the same position in the grid, the final numbers of the unsolved Sudoku correspond one-to-one with those of the auxiliary Sudoku.

(2) The average of the sum of the differences between the final numbers in the Sudoku to be solved and the corresponding numbers in the auxiliary Sudoku is N/2. When calculating each difference, a unified rule must be applied: either subtract the number in the Sudoku to be solved from the auxiliary Sudoku, or subtract the number in the auxiliary Sudoku from the Sudoku to be solved.

(3) The Sudoku puzzle to be solved must contain at least N−2 distinct clue numbers.

(4) In the same position of the grid, if the corresponding numbers of the Sudoku to be solved and the auxiliary Sudoku are the same, they will not be included in the calculation.

4. YYQ Algorithm for Solving the Unique Sudoku Problem

The YYQ algorithm fundamentally constructs a precursor auxiliary Sudoku without numerical constraints to solve the target Sudoku. When an N-cell Sudoku has a unique solution, the algorithm analyzes based on the parity of N: For even N, if the digit types in the final configurations of both the precursor and target Sudokus differ, the average of their digit difference sums equals N/2 according to Definition 2. For odd N, after discarding a specific digit type while maintaining distinct final configurations, the average of their digit difference sums remains N/2.

 doi:10.20944/preprints202507.0434.v3

Lemma 1: For a unique N-Grid Sudoku puzzle, when the types of numbers in the unsolved Sudoku differ from those in the auxiliary Sudoku, and the number of distinct types in the calculation is at least N−2, then a type of number in the auxiliary Sudoku corresponds to a type of number in the final unsolved Sudoku.

Proof: When the types of clue numbers in the auxiliary Sudoku differ from those in the target Sudoku, the corresponding numbers in the same grid position can be converted between the two. Given that the target Sudoku has a unique solution, its final configuration is entirely determined by the clue numbers and their positions. Consequently, the transformed clue numbers form a uniquely solvable Sudoku puzzle. Since the auxiliary Sudoku is uniquely determined, the transformed target Sudoku's final configuration matches the auxiliary Sudoku, allowing reverse deduction of the target Sudoku's solution. This demonstrates that when the clue numbers in the target Sudoku differ from those in the auxiliary Sudoku, each type of number in the auxiliary Sudoku corresponds to a specific type of number in the target Sudoku's final configuration. In such cases, the Sudoku-solving problem can be solved in polynomial time. However, this scenario is extremely rare in real-world applications and has limited practical significance.

It should be noted that when the number of missing digits in the Sudoku to be solved is N or N-1, the auxiliary Sudoku corresponds to a single solution for the Sudoku to be solved, maintaining a one-to-one correspondence between the two. When the number of missing digits is N-2, the auxiliary Sudoku corresponds to two solutions for the Sudoku to be solved, though the numerical correspondence between the two remains one-to-one. When the number of missing digits is N-3, the auxiliary Sudoku corresponds to multiple solutions for the Sudoku to be solved, and the numerical correspondence between the two may no longer be one-to-one. As the number of missing digits continues to decrease, the numerical correspondence between the auxiliary Sudoku and the Sudoku to be solved may also break one-to-one relationships. Therefore, to ensure that the final Sudoku to be solved and the auxiliary Sudoku maintain a one-to-one correspondence in terms of numerical types, the number of missing digits used in the calculation must be at least N-2.

Proposition 2: For a Sudoku with a unique solution in an N-grid, the number of possible clue digits must be at least N-1.

Proof: If the Sudoku puzzle contains fewer than N-1 distinct clue numbers, it must contain at least two non-identical numbers. By the principle of number substitution, these two numbers can be swapped, meaning the puzzle cannot have a unique solution. Therefore, for a Sudoku to be uniquely solvable, it must contain at least N-1 distinct clue numbers.

For a Sudoku puzzle with a unique solution, the number of possible clue digits must be at least N-1. Since the number of digits used in calculations must be at least N-2, and N-1 is clearly greater than N-2, the required number of digits can always be determined.

Lemma 3: For a Sudoku puzzle with a unique solution, excluding cases where the target numbers differ from their corresponding auxiliary numbers, the final target numbers in the auxiliary Sudoku corresponding to a specific type of auxiliary numbers will not be of the same type.

Proof: Under the proposition of Lemma 3, if the same-type numbers in the auxiliary Sudoku correspond to a specific type of numbers in the final state of the Sudoku to be solved, then this correspondence can only be guaranteed through swapping operations with other types of numbers in the auxiliary Sudoku. However, this condition is only valid under the proposition of Lemma 1, and clearly does not hold under the proposition of Lemma 3. Therefore, under the proposition of Lemma 3, the corresponding final-state numbers in the Sudoku to be solved must belong to different number categories.

The Sudoku puzzle numbers are divided into two groups based on their frequency: those with frequency 1 form one group, and those with frequency 2 form the other. The following outlines the detailed calculation process of the YYQ algorithm.

Theorem 1: For a unique N-sudoku, excluding cases where the target Sudoku's clue numbers differ from their corresponding auxiliary Sudoku numbers, the same-type digits in the auxiliary Sudoku will not all correspond to the same group of digits in the target Sudoku's empty cells.

 doi:10.20944/preprints202507.0434.v3

Moreover, each candidate number in the empty cells will contain at least one digit from the same group as the target number.

Proof: Since the final state of a Sudoku is entirely determined by the given numbers, this implies that the unsolved numbers in the remaining empty cells also influence the given numbers. As stated in Lemma 3, the types of unsolved numbers in the auxiliary Sudoku are not mutually exclusive. Only when the unsolved numbers in the remaining cells differ from the corresponding types in the auxiliary Sudoku, do the types of unsolved numbers in the auxiliary Sudoku become mutually exclusive. This allows us to extend Lemma 3's conclusion: after excluding the unsolved numbers, the unsolved cells corresponding to the same type of numbers in the auxiliary Sudoku will not contain identical numbers. According to Lemma 1, when numbers within the same group are swapped, the sum of differences between the auxiliary and unsolved Sudoku's final states remains equal to N/2. However, when numbers from different groups are swapped, this sum no longer equals N/2. To maintain the sum of differences at N/2, numbers from different groups cannot be swapped. Furthermore, if a candidate number in an unsolved cell is absent, it indicates that this absence is determined by the given numbers. In other words, the absence of a specific number in a candidate list is equivalent to the given number itself. This extends Lemma 3's conclusion: the unsolved cells corresponding to the same type of numbers in the auxiliary Sudoku will not all contain identical numbers from the same group, and each candidate list will include at least one number from the same group. Thus, the proof is complete.

5. Complexity Analysis of YYQ Algorithm for Solving the Unique Sudoku Problem

The construction process of pre-solver-assisted Sudoku employs a constant-time algorithm. By leveraging different digit permutations within the pre-solver-assisted Sudoku, the average of the sum of corresponding digit differences in the target Sudoku gradually converges to N/2. For instance, priority is given to swapping digit types with larger deviation differences. The execution time is clearly proportional to input scale, with a time complexity of O(N). The YYQ algorithm's time complexity is analyzed based on the digit arrangement in each row or column. In the worst-case scenario, the total time complexity for solving a Sudoku is (N^N×N^N). Whether dealing with odd or even grid Sudoku, one YYQ algorithm iteration can eliminate ($N^{N/2}\times N^{N/2}$) digit permutations. Since there exists only one valid solution, (N^N×N^N-1) digit combinations need to be eliminated. Eliminating (N^N×N^N-1) combinations requires (N^4-1) steps, resulting in an overall time complexity of O(N^4).

Since the time complexity of constructing a pre-solver-assisted Sudoku is constant, and the time complexity of constructing an auxiliary Sudoku from a pre-solver-assisted Sudoku is O(N), the YYQ algorithm's time complexity for solving a Sudoku is O(N) * O(N^4) = O(N^5). For an N-cell Sudoku with input size N^2, the time complexity is O(N^2.5). Therefore, the YYQ algorithm can always solve unique Sudoku problems in polynomial time.

Theorem 2 There exists a polynomial time algorithm to solve the Sudoku problem.

6. Conclusions

The algorithm proposed in this paper is applicable to solving Sudoku puzzles of any size, and its time complexity remains polynomial as the problem size N increases. For the N-grid Sudoku problem, this algorithm demonstrates that polynomial-time solutions exist for Sudoku, thereby providing empirical evidence for the existence of polynomial-time algorithms for NP-class problems, and ultimately proving P=NP.

References

1. Cook S A. The complexity of theorem-proving procedures[M]//Logic, automata, and computational complexity: The works of Stephen A. Cook. 2023: 143-152.

 doi:10.20944/preprints202507.0434.v3

2. Johnson D S, Garey M R. Computers and intractability: A guide to the theory of NP-completeness[M]. WH Freeman, 1979.
3. Yato T, Seta T. Complexity and completeness of finding another solution and its application to puzzles[J]. IEICE transactions on fundamentals of electronics, communications and computer sciences, 2003, 86(5): 1052-1060.
4. McGuire G, Tugemann B, Civario G. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration[J]. Experimental Mathematics, 2014, 23(2): 190-217.
5. Karp R M. Reducibility among combinatorial problems[M]//50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009: 219-241.
6. Lynce I, Ouaknine J. Sudoku as a SAT Problem[C]//AI&M. 2006.
7. Colbourn C J. The complexity of completing partial latin squares[J]. Discrete Applied Mathematics, 1984, 8(1): 25-30.
8. Utama F, Kridalaksana A H, Astuti I F. Implementasi Backtracking Algorithm Untuk Penyelesaian Permainan Su Doku Pola 9x9[J]. Inform Mulawarman J Ilm Ilmu Komput, 2016, 11(1): 29.
9. Knuth D E. Dancing links in Millennial Perspectives in Computer Science ed J Davies B Roscoe J Woodcock Eds[J]. 2000.
10. Gunther J, Moon T. Entropy minimization for solving Sudoku[J]. IEEE transactions on signal processing, 2011, 60(1): 508-513.
11. Arnold,Elizabeth,Lucas,et al.Gröbner Basis Representations of Sudoku.[J].College Mathematics Journal, 2010.
12. Nikita M.Sudoku Solving Algorithm and Grid Based Models for Digit Recognition[J].International Journal for Research in Applied Science and Engineering Technology, 2023.DOI:10.22214/ijraset.