Essay

A Polynomial Time Algorithm for Solving Sudoku Problems

Youqiang Yu *

Posted Date: 4 July 2025

doi: 10.20944/preprints202507.0434.v1

Keywords: Sudoku; NP-complete problem; discrete normal distribution; Polynomial time algorithm

 doi:10.20944/preprints202507.0434.v1

Essay

A Polynomial Time Algorithm for Solving Sudoku Problems

Yu Youqiang

Hongshizhaizi Town Central Health Center, Huadian City, Jilin City, Jilin Province; 2621061585@qq.com

Abstract

The P vs. NP equivalence problem, one of the seven major mathematical challenges of the millennium, continues to challenge researchers in mathematics and computer science. At its core, this question explores whether the complexity classes P and NP are equivalent. NPC problems, the most challenging subset of NP problems, can lead to the proof that P=NP if a polynomial-time algorithm is found for any of them. The Sudoku solving problem, a typical NPC problem, has yet to be solved using a polynomial-time algorithm. This paper introduces a polynomial-time algorithm for solving Sudoku, aiming to demonstrate that NPC problems have polynomial-time solutions, thereby proving P=NP. This algorithm is not only applicable to standard 9x9 Sudoku but can also be extended to any grid size, significantly reducing computational complexity compared to traditional enumeration methods and their derivatives. The findings ofthis study may hold significant theoretical implications for exploring the P=NP problem.

Keywords: Sudoku; NP-complete problem; discrete normal distribution; Polynomial time algorithm

1. Introduction

In the realm of computer science, the P vs. NP equivalence problem serves as a beacon for theoretical exploration [1], guiding researchers to delve into the essence of computational efficiency. This fundamental question not only delineates the boundaries of algorithmic capabilities but also touches on the core paradigms of computer science. Problems in the P class are efficiently solved within polynomial time, reflecting the efficiency of the computational process; whereas problems in the NP class can verify solutions within polynomial time, but their solving processes may not be efficient, posing a significant challenge in algorithm research.

As the most complex set of problems in NP, NPC problems push the boundaries of computational complexity to the extreme [2]. If a polynomial-time algorithm for NPC problems exists, the P=NP conjecture would be proven [3], which would have a transformative impact on computer science. Many complex problems could then be solved efficiently within polynomial time, significantly advancing global technological and economic development.

The Sudoku problem, which originated from the Latin squares studied by mathematician Euler in the 18th century, has been proven to be a typical NPC problem[4]. In the 9x9 grid, players must use logical reasoning to fill in the numbers 1-9, ensuring that each row, column, and box contains unique digits. Despite its simple rules, solving it is highly challenging——when the number of known numbers reaches the theoretical minimum (17)[5], the complexity of solving it often significantly increases.

In this paper, a general polynomial time algorithm for solving Sudoku problem is proposed, aiming to prove P=NP by solving this NPC problem. The algorithm is compatible with any size of Sudoku grid, and it may achieve a breakthrough in the efficiency of solving Sudoku.

 doi:10.20944/preprints202507.0434.v1

2. Background and Problem Introduction

Taking the standard 9x9 Sudoku as an example, this article will delve into the derivation process of the algorithm and its related conclusions. According to the basic rules of Sudoku, the number of hints significantly influences the difficulty of the puzzle: fewer hints generally require more logical reasoning steps, thus increasing the difficulty. It has been proven that theoretically, a 9x9 Sudoku requires at least 17 hints to ensure a unique solution.

If a Sudoku has a unique solution, the final state of the Sudoku is entirely determined by the given numbers.

Given a known solution to a Sudoku, if the grid positions of the numbers in both the known and the unknown Sudoku remain unchanged, and these numbers are matched one-to-one, and if the given Sudoku has the same numbers in the same positions as the unknown Sudoku, then the final answer of the known Sudoku must be the final answer of the unknown Sudoku. In a 9x9 Sudoku, the numbers used are from 1 to 9. This article uses modular 9 operations (numbers cycling within the range of 1-9) as the analytical framework: if there is a constant difference between the given numbers and the numbers in the unknown Sudoku, it can be inferred that all corresponding positions in the final states of the two Sudokus will also differ by this constant. In such cases, the answer to the unknown Sudoku can be directly obtained by adding this constant to the known final answer.

In a 9x9 Sudoku grid, if known Sudokus that maintain a constant difference from the given numbers can be found, where the differences include zero, the solution to the unsolved Sudoku can be obtained by adding a constant to the known numbers. In this special case, the time complexity of solving the Sudoku can be reduced to a constant level. Clearly, solving Sudoku puzzles under these specific conditions is within the realm of polynomial-time solvable problems. However, in practice, obtaining such known Sudokus is often challenging. It's worth noting that constructing Sudokus according to certain rules without any given numbers is relatively straightforward and also falls within the class of polynomial-time solvable problems.

Based on the differences between known and unsolved Sudoku puzzles, this paper categorizes Sudoku solving problems into two types. First, if the known Sudoku puzzle and the unsolved Sudoku puzzle have only minor differences in the hints, the numbers can be adjusted through digital substitution to match the positions and values of the hints in the 9x9 grid, allowing for a direct solution. Alternatively, if the hints of the unsolved Sudoku puzzle and the known Sudoku puzzle differ by a constant value, the solution can be directly obtained by adding this constant. Second, if the hints differ significantly, making it impossible to solve the unsolved Sudoku puzzle by simply swapping numbers or adding a constant, this falls under the second category. In the first category, the problem is polynomial-time solvable; however, in the second category, the first method fails. Therefore, proving that the second category also has a polynomial-time solution would establish that Sudoku solving problems are polynomial-time solvable, thus confirming P=NP. After rigorous analysis, this paper proposes a polynomial-time algorithm for solving Sudoku puzzles in the second category, which will be detailed in the following sections.

3. Derivation and Results of Polynomial Time Algorithm for Sudoku Problems in the Second Case

When discussing the second type of problem, at the same grid position, the numbers in the unsolved Sudoku and the known Sudoku often differ; if the numbers are the same, it indicates that the difference between the two is 9. Clearly, the difference between the two numbers is always distributed between 1 and 9. By cumulatively adding the probabilities of specific differences in descending order (the first term is the probability of a difference of 9, the second term is the sum of the probabilities of differences of 9 and 8, and so on, until the total probability of differences from 9 to 1 is 1), we can observe that the probability distribution of these nine sets of differences closely resembles the area under a discrete Gaussian distribution curve with

 doi:10.20944/preprints202507.0434.v1

positive independent variables, with a total probability of 1. Therefore, the probability corresponding to each specific difference can be calculated using the formula for standardizing normal distributions [6]. By using the difference between the hints in the unsolved Sudoku and the known Sudoku as the sampling sample, we can estimate the frequency of each difference in the overall Sudoku.

By synchronously performing increment or decrement operations (with a step size of 1) on the numbers in a known Sudoku, nine interlocking Sudokus can be generated, including the initial known Sudoku. These nine Sudokus ensure that the difference between any two numbers is always the same constant. To simplify the solution process, the probability of the difference between these nine Sudokus and the Sudoku to be solved being 9 can be calculated sequentially. This ensures that, based on the initial known Sudoku, a complete probability distribution of differences from 1 to 9 with respect to the Sudoku to be solved is obtained. During the calculation, the difference values between the Sudoku to be solved and the known Sudoku hints must follow a unified rule, and the nine known Sudokus must use the same addition or subtraction operations to calculate the difference. Using the standardization formula for the normal distribution and the normal distribution probability table, the probability range for a specific difference value of 9 can be derived. By inferring the population from samples, the overall probability of the known Sudoku having this difference value relative to the target Sudoku can be determined. Multiplying the total number of cells in the 9x9 Sudoku by this probability value, and rounding the result to the nearest whole number, gives the number of valid numbers in the known Sudoku under the condition of a difference of 9. This determines the total number of numbers in the entire 9x9 grid that meet this condition.

Based on the distribution of candidate numbers in each cell, it can be determined whether the difference between the initial known numbers and the number in that cell meets a specific difference value condition. However, the number of cells that meet this condition is usually higher than the actual value. Based on the candidate numbers in each cell of the unsolved Sudoku and the number of numbers that meet the specific difference value condition, 18 equations can be established as follows:

A1+B1+C1+D1+E1+F1+G1+H1+I1= the number of blanks with a difference value of 1 (1)
A2+B2+C2+D2+E2+F2+G2+H2+I2= the number of blanks with a difference value of 2 (2)

A3+B3+C3+D3+E3+F3+G3+H3+I3= the number of blanks with a difference value of 3 (3)
A4+B4+C4+D4+E4+F4+G4+H4+I4= the number of blanks with a difference value of 4 (4)

A5+B5+C5+D5+E5+F5+G5+H5+I5= the number of blanks with a difference value of 5 (5)
A6+B6+C6+D6+E6+F6+G6+H6+I6= the number of blanks with a difference value of 6 (6)

A7+B7+C7+D7+E7+F7+G7+H7+I7= the number of blanks with a difference value of 7 (7)
A8+B8+C8+D8+E8+F8+G8+H8+I8= the number of blanks with a difference value of 8 (8)

A9+B9+C9+D9+E9+F9+G9+H9+I9= the number of blanks with a difference value of 9 (9)

A1+A2+A3+A4+A5+A6+A7+A8+A9=A (10)

B1+B2+B3+B4+B5+B6+B7+B8+B9=B (11)

C1+C2+C3+C4+C5+C6+C7+C8+C9=C (12)

D1+D2+D3+D4+D5+D6+D7+D8+D9=D (13)

E1+E2+E3+E4+E5+E6+E7+E8+E9=E (14)

F1+F2+F3+F4+F5+F6+F7+F8+F9=F (15)

G1+G2+G3+G4+G5+G6+G7+G8+G9=G (16)

H1+H2+H3+H4+H5+H6+H7+H8+H9=H (17)

I1+I2+I3+I4+I5+I6+I7+I8+I9=I (18)

A represents the number of digits that match a difference of 1, B represents the number of digits that match a difference of 2, C represents the number of digits that match a difference of 3, D represents the number of digits that match a difference of 4, E represents the number of digits

 doi:10.20944/preprints202507.0434.v1

that match a difference of 5, F represents the number of digits that match a difference of 6, G represents the number of digits that match a difference of 7, H represents the number of digits that match a difference of 8, and I represents the number of digits that match a difference of 9.

Clearly, these 18 equations all belong to a system of linear equations with multiple variables, with the number of independent variables reaching up to 81. Given that constructing Sudoku puzzles without numerical constraints is relatively straightforward, new known Sudoku instances can be created. The difference in numbers between each newly constructed known Sudoku and the original known Sudoku can be precisely calculated. Consequently, the new column equations are interrelated with the original 18 equations and form a system of linearly independent equations. Using this method, several more equations can be constructed, for example, by using 8 new Sudoku puzzles to create 8 systems, each containing 18 equations, and then applying Gaussian elimination to solve the system of linear equations. It has been proven that when the number of linearly independent equations is at least as many as the number of independent variables, solving the system of linear equations using Gaussian elimination is a polynomial-time complexity algorithm [7].

After obtaining the values from A1 to I9, assign unit values to the independent variables of each equation based on the differences. Specifically, assign the values 1 to 9 to A, B, C, D, E, F, G, H, and I, respectively, and then multiply these values by A1 to I9 in sequence. This process yields a set of values that match the number of difference cells in each equation. Next, label the numbers in the unsolved Sudoku as variables X1 to X81. Based on formulas (1) to (9), establish 9 equations for the specific cell positions where the known Sudoku matches the specific difference. Similarly, based on another 8 Sudokus, 72 equations can be established. Then, use Gaussian elimination to solve for the values of X1 to X81 sequentially. Finally, using the difference calculation results between the unknown Sudoku and the known Sudoku, solve the unsolved Sudoku.

4. Conclusions

The algorithm proposed in this study can solve Sudoku problems of any size, and its time complexity remains polynomial as the problem size n increases. For n × n Sudoku problems, the algorithm has the following characteristics: when the clues differ little from the known Sudoku, it can be solved quickly through a constant-time transformation; even when the clues differ significantly, the algorithm's time complexity still meets the definition of a polynomial-time algorithm. If this algorithm holds true, it indicates that there is a polynomial-time solution to Sudoku problems, providing empirical evidence for the existence of polynomial-time solutions to NP-class problems, and thus supporting the hypothesis that NP = P.

References

1. THE CLAY MATHEMATICS INSTITUTE. P vs NP Problem [EB/OL] .https:// www.claymath.org/millennium-p roblems/ p-vs-np-problem.
2. COOK S A. The complexity of theorem-proving procedures [C]// Proceedings of the third annual ACM symposium on Theory of computing. 1971: 151-158.
3. GAREY M R, JOHNSON D S. Computers and intractability: a guide to the theory of NP-completeness [M]. San Francisco: W. H. Freeman, 1979
4. YATO T, SETA T. Complexity and completeness of finding another solution and its application to puzzles
5. MCGUIRE G, TUGEMANN B, CIVARIO G. There is no 16-clue sudoku: solving the sudoku minimum number of clues problem [J]. Experimental Mathematics, 2014, 23(2): 190-217.
6. Chen Xiru. A Brief History of Mathematical Statistics [M]. Changsha: Hunan Education Press, 2000: 28–156.

 doi:10.20944/preprints202507.0434.v1

7. EDMONDS J.Systems of linear equations[M]// Lectures on Algebraic Algorithms.Berkeley: Mathematical Sciences Research Institute, 1967: 1–20.