Computational Complexity of Sudoku

Thomas Varvella Vicente

December 2023

1 Problem Statement

Generalized Sudoku: Sudoku is a well-known puzzle game typically played on a 9x9 grid, divided into 3x3 subgrids. The objective is to fill the grid with digits so that each column, each row, and each of the nine 3x3 subgrids contain all of the digits from 1 to 9. A generalized version of Sudoku extends this concept to an $n \times n$ grid where each row, column, and designated subgrid (of a size determined by the rules of the puzzle) must contain all the digits from 1 to n without repetition. This generalization can include variations in grid size, subgrid configuration, and initial clues, thereby introducing a range of complexities and challenges beyond the traditional 9x9 Sudoku.

2 Proof that Generalized Sudoku is in NP

To prove that Generalized Sudoku is in NP, we must demonstrate that a given solution can be verified in polynomial time. A solution for a Generalized Sudoku puzzle involves filling an $n \times n$ grid with integers such that each row, column, and designated subgrid contains all integers from 1 to n without repetition.

The verification algorithm proceeds as follows: for each row, column, and subgrid, check if all integers from 1 to n appear exactly once. This process is inherently polynomial, as it requires $O(n^2)$ checks for each row and column, and a similar number of checks for each subgrid.

2.1 Pseudocode for Verification

```
function verifySudokuSolution(grid):
    n = length(grid)
    for i in 1 to n:
        if not isUnique(grid[i, 1..n]) or not isUnique(grid[1..n, i]):
            return false
    for each subgrid in grid:
        if not isUnique(extractNumbers(subgrid)):
            return false
    return true
```

```
function isUnique(array):
    seen = set()
    for element in array:
        if element in seen:
            return false
        seen.add(element)
    return true
```

The function ‘verifySudokuSolution‘ checks each row, column, and subgrid for the uniqueness of numbers. The ‘isUnique‘ function helps in this process by ensuring that each number appears only once. The verification, therefore, can be done efficiently, confirming that Generalized Sudoku belongs to the NP class. Additionally, it's important to note that the size of the witness (the filled

$n \times n$ grid) is polynomially related to the size of the instance (the puzzle itself). In a Generalized Sudoku puzzle, the witness is the complete grid, which has n^2 entries. Since the puzzle size is also determined by n, the size of the witness is a polynomial function of the size of the instance, further confirming that Generalized Sudoku belongs to NP.

3 Introduction to Latin Squares and Related Problems

Latin squares are $n \times n$ grids filled with n different symbols, each occurring exactly once in each row and column. Originating from mathematical puzzles, Latin squares are closely related to Sudoku and have significant applications in statistics and the design of experiments.

3.1 Latin Square Problems

The primary problems associated with Latin squares, as discussed in the provided literature, are:

- **Completing a Partial Latin Square:** Given a partially filled $n \times n$ grid with some symbols already placed, the challenge is to fill in the remaining cells without violating the Latin square conditions.
- **Latin Square Property Verification:** Verifying whether a given $n \times n$ grid is a Latin square involves checking that each symbol appears exactly once in each row and column.

A Latin square of order n is an arrangement of n symbols in an $n \times n$ grid, such that each symbol appears exactly once in each row and each column. Mathematically, it can be represented as an $n \times n$ matrix where each cell contains one of n symbols, and no symbol is repeated in any row or column. This property of

non-repetition makes Latin squares an excellent model for certain combinatorial problems and their graphical representations, where vertices represent cells, and edges connect cells in the same row or column. The structure of the graph derived from a Latin square highlights the interplay between different rows and columns, underpinning the complexity of Latin square problems.

4 NP-completeness of SUDOKU

Based on the work of Paul W. Goldberg

This result was first shown in a master's thesis by reduction from the NP-complete problem LATIN SQUARE COMPLETION. The reduction is direct and demonstrates the complexity of Sudoku.

4.1 Latin Square Completion

A Latin square of order n is an $n \times n$ grid that contains n occurrences of each number in the range $1-n$, such that no row or column contains two occurrences of the same number. LATIN SQUARE COMPLETION involves filling in the blank entries of a partially filled Latin square to obtain a complete Latin square.

4.2 Example with $n = 4$

Consider a 4×4 instance of LATIN SQUARE COMPLETION as follows:

4			3
	2		
		4	2
1			

4.3 Corresponding Sudoku Instance

To encode this into a Sudoku instance, we copy each column of the LATIN SQUARE COMPLETION instance into the first columns of the top row of squares in the Sudoku instance, and fill up the rest with other numbers. Here is the corresponding 16×16 Sudoku instance:

4	5	9	13		8	12	16		7	11	15	3	6	10	14
	6	10	14	2	5	9	13		8	12	16		7	11	15
	7	11	15		6	10	14	4	5	9	13	2	8	12	16
1	8	12	16		7	11	15		6	10	14		5	9	13

(Goldberg 2023)

The remaining cells are filled with numbers ranging from 5 to 16 in such a way that the problem of completing the Sudoku is equivalent to completing the original Latin square.

4.4 Polynomial time Computability

The general rule for constructing a $n^2 \times n^2$ instance of Sudoku from an $n \times n$ instance of LATIN SQUARE COMPLETION involves ensuring that there is a polynomial-time computable way of filling in the idle squares with the numbers $n + 1$ through n^2. This ensures the equivalence of the complexity between the two problems.

5 Reduction from Tripartite Graph Triangulation to Latin Square Completion

Based on the works of Ruby(Ziheng) Ru, Kristen(Zidan) Huang, Lingge Nox Yu

The problem of triangulating a tripartite graph can be reduced to the problem of completing a Latin square. This section outlines the process of this reduction and illustrates the key concepts with graphics.

5.1 Tripartite Graph Triangulation

A tripartite graph is a special type of graph where the set of vertices can be divided into three disjoint sets such that no two vertices within the same set are adjacent. The goal of triangulation is to add a minimum number of edges to make the graph triangulated, meaning that every face in the graph (including the outer face) is a triangle.

5.2 graph triangulation

Graph triangulation is a process applied to undirected graphs where additional edges are added to ensure that every face (including the outer face) of the graph is bounded by a triangle. This transformation is crucial in various fields such as computational geometry, graph theory, and network analysis. In triangulation, the primary goal is to convert a given graph into a chordal graph, a type of graph where every cycle of four or more vertices has a chord, which is an edge that is not part of the cycle but connects two vertices of the cycle. The significance of triangulating a graph lies in simplifying the structure for certain types of analysis, such as finding the maximum clique, graph coloring, or solving network flow problems. The method of triangulation varies based on the

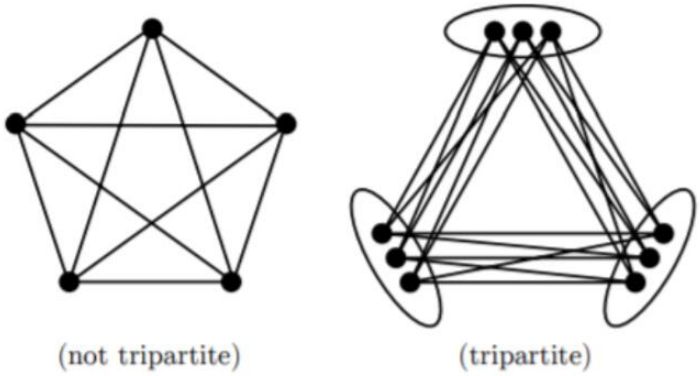

Figure 1: a tripartite graph and a non-tripartite graph (Ru, Huang, Yu)

type of graph and the specific requirements of the problem at hand. In some contexts, particularly in mesh generation and geographical information systems, triangulation is used to approximate surfaces with triangles, leading to efficient computational models.

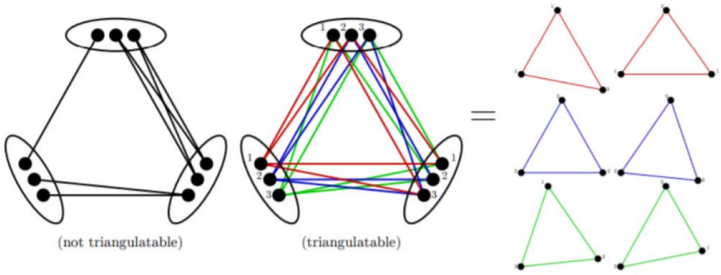

Figure 2: This is a visual of a triangulated graph (Ru, Huang, Yu)

5.3 Reduction Process

To demonstrate the reduction of the problem of triangulating a tripartite graph (with vertex sets X, Y, Z) to that of completing a Latin square, we consider a step-by-step approach:

1. **Vertex Set Representation:** Begin by representing each of the three vertex sets X, Y, Z of the tripartite graph as one of the dimensions in a Latin square of size $n \times n$, where n is the number of vertices in each set. Each cell in the Latin square is thus denoted by a tuple (x, y, z), where $x \in X$, $y \in Y$, and $z \in Z$.

2. **Edge Mapping:** For each edge connecting vertices in the tripartite graph (e.g., an edge between vertex x in X and vertex y in Y), identify the corresponding cell in the Latin square and fill it with the appropriate value. The value filled is the corresponding vertex from the third set Z that is not directly connected by this edge. This creates a partial filling of the Latin square.

3. **Triangulation and Completion:** The objective is to triangulate the tripartite graph by adding the minimum number of edges. In the Latin square, this translates to adding entries such that each row, column, and symbol (representing each vertex in Z) appears exactly once. The completion of the Latin square thus corresponds to a triangulated tripartite graph where each added edge in the graph fills a missing cell in the square.

5.4 Visualization of the Reduction

Imagine the following instance of a partially completed Latin square exists, where any cells that have no color associated are part of the pre-filling of the square. The other cells can be filled in according to the triangulation of the corresponding tripartite graph. To connect this image with the previous subsection about the reduction process, we can say that for any tuple (x, y, z), where $x \in X$ corresponds to a row, $y \in Y$ corresponds to a column, and $z \in Z$ corresponds to the symbol being used. This way, we can ensure a valid instance of completion of the pre-filled Latin square from the instance of a triangulated tripartite graph.

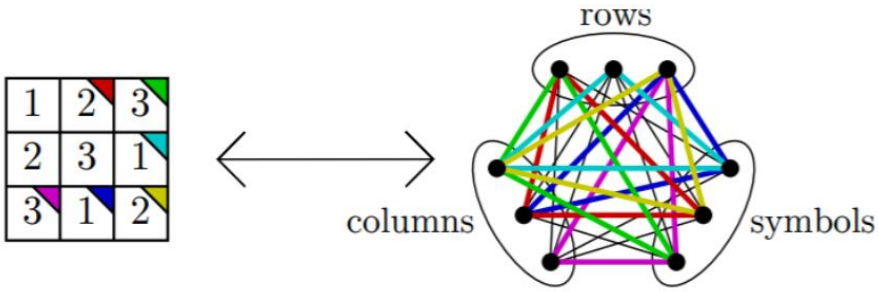

Figure 3: This illustration depicts the detailed reduction process from triangulating a tripartite graph to Latin square completion. It includes an example of a tripartite graph and its corresponding partially filled Latin square, highlighting how edges in the graph map to entries in the square, and how triangulating the graph equates to completing the square. (Ru, Huang, Yu)

This reduction shows the equivalence of these two problems and provides a novel way to approach the problem of tripartite graph triangulation by leveraging methods used in solving Latin square completion problems.

5.5 Polynomial Time Complexity of the Reduction

The reduction from triangulating a tripartite graph to completing a Latin square can be achieved in polynomial time, which is crucial for the complexity analysis of the problem. The key steps in establishing this polynomial time computability are as follows:

1. **Initial Mapping:** The initial step involves mapping the vertices of the tripartite graph to the dimensions of the Latin square. This is a straightforward process that involves creating a $n \times n$ grid, where n is the number of vertices in each vertex set of the tripartite graph. This step is linear with respect to the size of the vertex sets, and thus can be done in $O(n)$ time.

2. **Edge Representation:** Mapping each edge of the tripartite graph to a corresponding cell in the Latin square is also a polynomial-time process. Given that the total number of edges in a tripartite graph is bounded by $O(n^2)$, iterating over these edges to fill in the Latin square requires at most $O(n^2)$ operations.

3. **Checking for Triangulation:** Determining whether the Latin square is complete (and thus whether the tripartite graph is triangulated) involves checking each row, column, and the presence of each symbol from the third vertex set exactly once in the square. This validation can be performed in $O(n^2)$ time, as it requires a constant number of operations per cell.

4. **Edge Addition for Completion:** In the worst case, filling the missing entries in the Latin square to achieve triangulation may require examining each cell of the square. However, this process remains polynomial, specifically $O(n^2)$, as it involves a bounded number of operations for each cell.

Thus, the overall reduction process is polynomial in the size of the input tripartite graph, indicating that the triangulation problem's computational complexity is preserved during the reduction to the Latin square completion problem.

5.6 Instance of the reduction from triangulated tripartite to Latin square

Given the complexity of translating a triangulated tripartite graph derived from the 3SAT formula $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$ into a Latin square, we undertake a step-by-step analysis:

5.6.1 Initial Graph and Latin Square Setup

- We start with a triangulated tripartite graph resulting from the 3SAT formula, consisting of interconnected $H_{3,n}$ graphs for each variable and clause.

- The Latin square is set up to represent the tripartite graph, where each dimension (rows, columns, and symbols) represents one set of vertices. The sets are denoted as X, Y, and Z.

5.6.2 Mapping Graph Vertices to Latin Square

- Each vertex in the tripartite graph is mapped to a specific element in the Latin square. For instance, vertices in set X correspond to rows, set Y to columns, and set Z to symbols.

5.6.3 Translating Triangulation into Latin Square Entries

- Edges in the triangulated graph are translated into filled cells in the Latin square. For an edge between vertices representing row x and column y in the graph, the cell (x, y) in the Latin square is filled with the symbol corresponding to the third vertex in the triangle.
- This mapping ensures that each symbol in set Z appears exactly once in each row and column of the Latin square, adhering to its defining rules.

5.6.4 Completing the Latin Square

- Incomplete parts of the Latin square are filled following the Latin square rules: each symbol appears exactly once in every row and column.
- The process of filling these cells mirrors the addition of edges in the tripartite graph to achieve full triangulation.

5.6.5 Final Verification

- The completed Latin square is checked to ensure compliance with the rules of Latin squares, confirming that each symbol appears exactly once in each row and column.
- The successful completion of the Latin square reflects the triangulation of the tripartite graph, which in turn corresponds to a satisfying assignment for the 3SAT formula.

6 Reduction from 3SAT to Triangulating Tripartite Graphs

Based on the works of Padraic Bartlett

The problem of 3-SAT can be reduced to the problem of triangulating a tripartite graph.

To establish this reduction, we first delve into a topological analogy involving the transformation of a square of flexible material into a torus, analogous to graph gluing

Graph Gluing: Two graphs G_1 and G_2 can be glued together along a common sub-graph H by identifying the vertices and edges of H in both graphs. This process forms a combined graph where H acts as a junction.

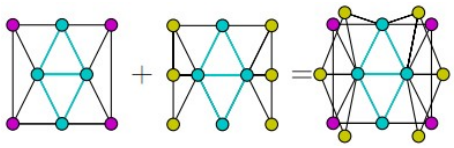

Figure 4: two graphs can be glued together through their shared sub-graph (Bartlett)

Construction of $H_{3,n}$: We construct the graph $H_{3,n}$ by gluing n copies of a specific triangular graph structure and then applying a toroidal transformation, where opposite edges of the resultant lattice are identified (glued) to form a triangulated torus.

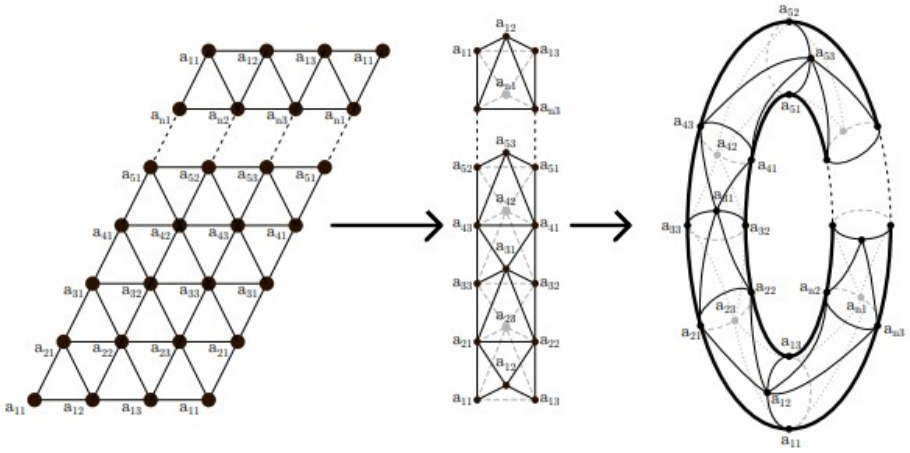

Figure 5: taking a triangular lattice and turning it into a toroidal object (Bartlett)

6.1 Triangulation and Boolean Logic

The triangulation properties of $H_{3,n}$ graphs, for $n > 3$, are central to their correspondence with boolean logic in the context of reducing 3SAT to graph triangulation. These graphs admit exactly two types of triangulations: 'true', using Δ (delta) triangles, and 'false', using ∇ (nabla) triangles. This dichotomy is key to interpreting boolean logic within the graph's structure.

6.1.1 Graph Gluing and Its Implications

The gluing of two $H_{3,n}$ graphs, especially along specific configurations known as A-patches and B-patches, significantly impacts the triangulation and the graph's relation to boolean formulas.

A-Patch and B-Patch in Gluing:

- Gluing an A-patch from one graph to an A-patch of another, after removing the central triangle from both, forces one graph to adopt a false triangulation and the other a true triangulation.
- Gluing an A-patch to a B-patch results in either both graphs having false triangulations or one having a true triangulation and the other a false one.

6.1.2 Construction of Graphs from 3SAT Formulas

The reduction process involves creating $H_{3,n}$ graphs for each variable x_k (denoted as C_{x_k}) and each literal $l_{i,j}$ (denoted as $C_{i,j}$) in a 3SAT formula. These graphs are then interconnected in a manner that reflects the formula's clauses and the nature of the literals, using A-patches for positive literals and B-patches for negated ones. In the case of clauses, the central triangle in the A-patches is removed to enforce the predetermined triangulation pattern.

6.1.3 Triangulatability and Satisfiability

The triangulatability of the final graph structure directly mirrors the satisfiability of the 3SAT formula. A satisfiable formula corresponds to a feasible triangulation pattern across the interconnected $H_{3,n}$ graphs, following the 'true' and 'false' triangulation principles. Conversely, an unsatisfiable formula results in a graph that cannot be triangulated while adhering to the constraints of the A-patches and B-patches.

This process encapsulates the reduction from 3SAT to triangulating tripartite graphs, where the triangulatability of the constructed graph becomes a representation of the boolean formula's satisfiability.

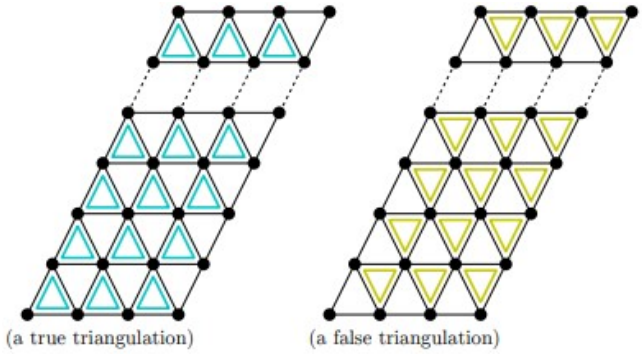

Figure 6: (Bartlett)

Thus, each boolean variable and literal in the 3SAT formula is represented by separate $H_{3,n}$ graphs. The specific gluing operations, based on the 3SAT formula's structure, ensure that a valid triangulation of the entire interconnected graph structure corresponds to a satisfying assignment of the boolean formula.

6.2 Completing the Reduction

The final step in the reduction is to demonstrate that a triangulation of the constructed graph implies a satisfiable boolean formula, and vice versa. This is achieved by correlating the 'true' and 'false' triangulations of individual $H_{3,n}$ components with the truth values of the corresponding boolean variables and literals.

Thus, the problem of determining a satisfying assignment for a 3SAT formula is effectively reduced to triangulating a specially constructed tripartite graph. This establishes the polynomial-time reducibility of 3SAT to the problem of graph triangulation.

6.3 Polynomial Time Complexity of the Reduction

The reduction from 3SAT to the triangulation of a tripartite graph can be shown to be executable in polynomial time. The key steps in the reduction process involve constructing a specific type of graph representation for the 3SAT instances and then analyzing the triangulation possibilities. Here, we examine each step of the reduction process and demonstrate its polynomial time complexity.

1. **Graph Construction for 3SAT Instances:** For each variable and clause in the 3SAT formula, a corresponding H3,n graph is constructed. The number of these graphs is linearly proportional to the number of variables and clauses in the 3SAT instance. Creating each H3,n graph requires a number of steps linear in n, the size parameter of the graph.

2. **Gluing Operations:** The gluing of graphs along specific subgraphs (A-patches or B-patches) corresponds to the occurrences of variables and their negations in the clauses. This process is also polynomial in nature as it involves a fixed number of operations per variable and clause in the 3SAT instance.

3. **Triangulation Analysis:** The final step involves analyzing the triangulation of the constructed graph. This analysis is based on the structure of the graph which is directly derived from the 3SAT instance. Since the graph's size and the number of potential triangulations are polynomially bounded by the size of the 3SAT instance, this step also falls within polynomial time complexity.

In conclusion, each step of the reduction process is executable in polynomial time with respect to the size of the 3SAT instance. Therefore, the overall reduction process from 3SAT to triangulating a tripartite graph is polynomial-time computable.

6.4 Reduction from 3SAT to Tripartite Graph Triangulation

In this section, we illustrate the reduction of the 3SAT instance $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$ to the problem of triangulating a tripartite graph.

6.4.1 Initial Graph Construction

Step 1: Constructing Base Graphs for Variables.

- Construct base H3,n graphs for each unique variable in the formula. In this case, we have three variables: x, y, z. Each variable is represented by a distinct H3,n graph, which is a triangular lattice structure.
- Label these graphs as Cx, Cy, Cz respectively.

Step 2: Constructing Clause Graphs.

- For each clause in the formula, construct a separate H3,n graph. These graphs also have a triangular lattice structure.
- The first clause $(x \vee y \vee \neg z)$ results in a graph labeled $C_{clause1}$, and the second clause $(\neg x \vee \neg y \vee z)$ in $C_{clause2}$.

6.4.2 Gluing Process

Step 3: Gluing Clause Graphs to Variable Graphs.

- For each literal in the clauses, identify the corresponding variable graph. If the literal is positive, glue the clause graph to the 'A-patch' of the variable graph. If the literal is negative, glue it to the 'B-patch'.
- In $C_{clause1}$, glue:
  - x to the A-patch of Cx,
  - y to the A-patch of Cy,
  - $\neg z$ to the B-patch of Cz.
- In $C_{clause2}$, glue:
  - $\neg x$ to the B-patch of Cx,
  - $\neg y$ to the B-patch of Cy,
  - z to the A-patch of Cz.

6.4.3 Final Graph Assembly

Step 4: Assembling the Final Graph.

- Combine all the individual graphs (variable and clause graphs) into one large graph.
- This step involves merging vertices and edges at the points where the graphs have been glued together.

6.4.4 Triangulatability Analysis

Step 5: Analyzing Triangulatability.

- The final graph's triangulatability corresponds to the satisfiability of the original 3SAT formula.
- A triangulation of this graph implies a satisfying assignment for the formula $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$.

This reduction illustrates how the complex problem of 3SAT can be transformed into a graph-theoretic problem. The process involves constructing and gluing graphs in a way that encapsulates the logical structure of the formula, ultimately leading to a graph whose triangulatability reflects the satisfiability of the original 3SAT instance.

7 References

1. Discussing NP-Completeness of Sudoku Game: https://jcrouser.github.io/CSC250/projects/sudoku.html
2. NP-completeness of SUDOKU: https://www.cs.ox.ac.uk/people/paul.goldberg/FCS/sudoku.htmlhttps://
3. Minilecture 7: 3SAT and Latin Squares:
https://web.math.ucsb.edu/ padraic/ucsb$_2$013$_1$4/$mathcs103_s2014/mathcs103_s2014_m lecture7.pdf$