國立臺北科技大學
National Taipei University of Technology

資訊工程系碩士班

碩士學位論文

Department of Computer Science &
Information Engineering
Master Thesis

A Hybrid Constraint Propagation and Ant Colony System for Solving Three Dimensional Sudoku

研究生：楊昌澔

Researcher: Yang, Chang-Hao

指導教授：陳香君博士

Advisor: Shiang-Jiun Chen, Ph.D.

June 2026

國立臺北科技大學
研究所碩士學位論文口試委員會審定書

本校 資訊工程 研究所 楊昌澔 君

所提論文，經本委員會審定通過，合於碩士資格，特此證明。

學位考試委員會

委　員：葛宜学

馬良蔵

陳香君

指導教授：陳香君

所　長：劉建宏

中 華 民 國　一百一十五　年　六　月　十　日

Abstract

Keyword: Three Dimensional Sudoku, Constraint Propagation, Ant Colony System, Constraint Satisfaction Problem

Three dimensional Sudoku generalizes the classic 9×9 puzzle into a $9 \times 9 \times 9$ cube, introducing constraints across six unit types and substantially enlarging the search space. While prior work on 3D Sudoku has focused primarily on deterministic methods such as backtracking, the potential of combining constraint propagation with swarm intelligence has not been explored.

This thesis proposes a hybrid solver that applies Constraint Propagation (CP) first and invokes Ant Colony System (ACS) only when the puzzle cannot be completed by CP alone. Under this design, CP performs deterministic reduction of the search space, while ACS explores the residual space that remains after propagation.

The framework is evaluated through three experiments: a public benchmark test on 5,354 instances, a clue-density stress test on generated puzzles, and a sparse residual-search comparison between CP with ACS and a deterministic CP with DFS baseline. On the public benchmark, CP alone solved 99.70% of all instances with a mean runtime of 4.135 ms, while ACS completed the remaining 16 CP-unsolved instances. The clue-density stress test covered densities from 10% to 80%, where CP handled all instances at densities of 20% and above, while performance degraded at the sparsest density of 10%. In the additional 1%–10% sparse comparison, CP with DFS outperformed CP with ACS on most generated instances, indicating that deterministic DFS is a stronger residual-search baseline for current 9×9×9 puzzles. Nevertheless, ACS showed limited evidence of helping in rare sparse hard cases where deterministic search failed within the time limit.

TAIPEI TECH

Acknowledgements

本論文能夠順利完成，首先要誠摯感謝我的指導教授陳香君博士。在研究過程中，老師於研究方向、論文架構、實驗設計與寫作修正上給予我許多寶貴的建議與指導，使我能逐步釐清研究問題，並完成本論文。老師細心的研究態度與耐心的指導，讓我在碩士求學期間受益良多。

同時，感謝各位口試委員於論文口試期間提供專業且細緻的建議，使本論文能夠更加完整與嚴謹。委員們的指正與建議，對本研究的修正與完善有很大的幫助。

此外，感謝國立臺北科技大學與資訊工程系碩士班提供良好的學習環境與研究資源，使我能在研究所期間持續學習、探索並完成本論文。

也感謝研究所期間曾經協助我的同學、朋友與實驗室夥伴。在課業、研究與生活中，因為有你們的討論、鼓勵與陪伴，使這段研究歷程不再孤單，也讓我在遇到困難時能夠持續前進。

我也要特別感謝我的家人。謝謝你們在我求學期間給予支持、理解與鼓勵，讓我能夠專心完成學業與研究。你們的陪伴與支持，是我完成這段旅程的重要力量。

最後，感謝這段碩士研究歷程中所有曾經幫助過我的人。完成本論文不僅代表一個研究階段的結束，也象徵著新的開始。謹以此論文獻給所有曾經支持、鼓勵與幫助我的人。

楊昌澔 謹誌於

國立臺北科技大學資訊工程系碩士班

中華民國一一五年六月十號

TAIPEI TECH

Table of Contents

Abstract i

Acknowledgements ii

Table of Contents iii

List of Figures v

List of Tables vi

Chapter 1 Introduction 1

Chapter 2 Related Work 4

2.1 Problem Definition and Complexity 4

2.1.1 Problem Definition 4

2.1.2 Computational Complexity and NP-Completeness 5

2.2 Constraint Propagation for Sudoku 6

2.2.1 Sudoku as a CSP and the Effectiveness of Propagation 7

2.2.2 Constraint Propagation Techniques 7

2.3 Evolutionary and Metaheuristic Algorithms for Sudoku 7

2.3.1 Genetic Algorithm and Particle Swarm Optimization 8

2.3.2 Other Metaheuristic Methods 9

2.4 Ant Colony Optimization Framework 10

2.4.1 Application of ACO to Constraint Satisfaction Problems 10

2.4.2 From ACO to AS and ACS 11

2.4.3 Application of Ant Colony Optimization to Sudoku 12

2.5 Current Researches on 3D Sudoku 13

Chapter 3 Proposed Algorithm 15

3.1 Main Method 15

3.2 Implementation Details 15

3.2.1 Requirements 16

3.2.2 Implementation Steps 17

Chapter 4 Result and Analysis 20

TAIPEI TECH

4.1 Experimental Design 20
4.2 Public Benchmark Evaluation 20
4.2.1 Overall benchmark categorization 20
4.2.2 Summary statistics of CP-only solved instances 21
4.2.3 Detailed results of instances requiring ACS 21
4.3 Clue-Density Stress Test under 10%–80% Clue Densities 22
4.3.1 Motivation and dataset construction 22
4.3.2 Summary results across clue-density levels 22
4.3.3 Solved-ratio analysis 23
4.3.4 Runtime analysis 23
4.3.5 Residual-space analysis after CP 24
4.4 Sparse Residual-Search Comparison between ACS-Based and DFS-Based Search 25
4.4.1 Motivation and experimental setup 25
4.4.2 Solved count comparison under 1%–10% clue densities 26
4.4.3 Behavioral comparison of ACS-based and DFS-based residual search . 27
4.4.4 A rare sparse case where ACS succeeds when DFS fails 28
Chapter 5 Conclusion and Future Work 30
5.1 Conclusion 30
5.2 Future Work 31
References 33

TAIPEI TECH

List of Figures

1.1 An example of a 2 × 2 × 2 three-dimensional Sudoku. 2

2.1 3D Sudoku connectivity graph of 27 sub cubes (S1−S27). 5
2.2 All elements inside sub cube S1. 6

3.1 Flowchart for the Hybrid Framework of CP and Ant Colony System for 3D Sudoku 16

4.1 CP-only and overall solved ratios under different clue densities. 23
4.2 Mean runtime under different clue densities. 24
4.3 Mean number of remaining cells after CP under different clue densities. 25

TAIPEI TECH

List of Tables

2.1 Comparison between Ant System (AS) and Ant Colony System (ACS) 12

3.1 Software requirements 16
3.2 Python package requirements 17
3.3 Hardware requirements 17

4.1 Overall benchmark categorization 21
4.2 Summary statistics for CP-only solved instances 21
4.3 Detailed results for the 16 ACS-required instances 22
4.4 Clue-density stress-test generation summary 22
4.5 Solved count comparison under 1%–10% clue densities 26
4.6 A rare sparse case where ACS succeeds when DFS fails 28

TAIPEI TECH

Chapter 1 Introduction

Sudoku, a logic based number puzzle, gained global popularity after being introduced as "Number Place" by Howard Garns in the United States and later named "Sudoku" by Nikoli in Japan. Its widespread presence in newspapers, mobile apps, and puzzle books signifies its enduring appeal.

The inherent combinatorial structure and complexity of Sudoku attracted significant academic interest, extending its application beyond mere entertainment into information security, cryptography, cognitive science, and education. Recent research leverages Sudoku matrices for data hiding and steganography [1, 2], offering enhanced encoding security. Additionally, Sudoku's complex logic and symmetry support novel key generation mechanisms [3, 4, 5] for encryption algorithms. Cognitive studies demonstrate Sudoku's utility in strengthening pattern recognition and logical reasoning, as indicated by functional neuroimaging and randomized trials [6].

The emergence of three dimensional (3D) Sudoku expands its practical applications, further increasing data embedding capacity in steganographic and cryptographic fields. The 3D variant generalizes the classic 9×9 grid into a 9×9×9 cube, requiring each axis aligned line to contain digits 1 through 9 once, and each 3×3×3 subcube to hold three copies of each digit. To generalize, in an $n \times n \times n$ three dimensional Sudoku, the cube is divided into $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ subcubes, each of size $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$. Within each subcube, every digit $d \in \{1, 2, \ldots, n\}$ appears exactly $\sqrt{n}$ times, and along any axis aligned line, each digit occurs exactly once. As shown in Figure 1.1, the 2 × 2 × 2 3D Sudoku demonstrates the arrangement of digits within the cube.

Sudoku is classified as an NP-complete (Nondeterministic Polynomial-time Complete) problem [7], commonly serving as a benchmark in constraint satisfaction [8], SAT [9, 10], and optimization algorithm research [11]. Metaheuristics such as ant colony optimization (ACO) [12], genetic algorithms (GA) [13], and particle swarm optimization (PSO) [14] yield effective results in 2D Sudoku [15]. In contrast, most 3D Sudoku solution methods remain brute force or backtracking based [16, 17], while the use of swarm intelligence for 3D Sudoku remains

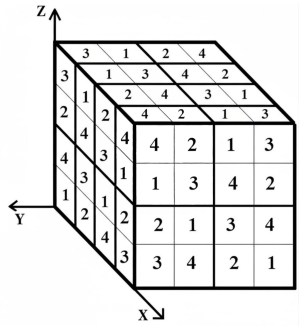


Figure 1.1: An example of a 2 × 2 × 2 three-dimensional Sudoku.

relatively limited.

This research gap motivates this study. Building on prior ACS based Sudoku solving frameworks, this thesis extends the idea from the conventional 2D setting to 3D Sudoku and proposes a hybrid framework that applies Constraint Propagation (CP) first and then invokes Ant Colony System (ACS) when the puzzle cannot be completed by CP alone. Under this design, CP is responsible for early deterministic reduction of the search space, whereas ACS is used to further explore the residual search space that remains after propagation.

Accordingly, the goal of this study is to investigate the respective roles of CP and ACS within a unified 3D Sudoku solving framework. We aim to examine whether CP can efficiently solve most public benchmark instances, and to compare ACS with DFS as residual-search strategies when CP alone cannot complete the puzzle. In addition, this study also investigates how the behavior of the proposed framework changes under different clue density settings, so that the strengths and limitations of the hybrid design can be more clearly understood.

To achieve these objectives, the proposed framework is implemented and evaluated through a series of experiments. The experiments include evaluation on a public benchmark of 3D Sudoku instances [18], a clue-density stress test based on generated puzzles, and an additional sparse residual-search comparison between CP with ACS and CP with DFS. The results show that CP serves as the dominant solving mechanism on the benchmark. The additional comparison further shows that CP with DFS is a stronger baseline for most sparse 9×9×9 instances, while ACS may still provide limited stochastic search diversity in rare difficult cases.

The remainder of this thesis is organized as follows. Chapter 2 reviews the theoretical background and related studies, including Sudoku as a constraint satisfaction problem (CSP),

existing metaheuristic approaches, the development from ACO to ACS, and current research on 3D Sudoku. Chapter 3 presents the proposed hybrid framework and its implementation details. Chapter 4 reports the experimental design, evaluation results, and analysis. Finally, Chapter 5 summarizes the key findings and highlights opportunities for future work.



Chapter 2 Related Work

The purpose of this chapter is to review the related literature and establish the theoretical foundation of this study. First, we define the Sudoku problem from a computational standpoint and discuss its complexity. Second, we introduce Sudoku as a CSP and review CP as a classical and highly effective solving approach [19]. Third, we review the applications of metaheuristic algorithms in solving Sudoku [20]. Then, we focus on the development of ACO in CSPs and its applications in Sudoku. In particular, the relationship among ACO, Ant System (AS), and ACS is clarified, since ACS is the variant adopted in this study. Finally, we summarize the current research status of 3D Sudoku and identify the research gap that motivates the proposed hybrid CP–ACS framework.

2.1 Problem Definition and Complexity

2.1.1 Problem Definition

A standard N × N Sudoku problem can be viewed as a variation of a partially filled Latin Square. According to the Integer Linear Programming (ILP) model, the Sudoku problem can be defined as a set of variables, domains, and constraints.

Let $X_{i,j}$ be the value in the cell located at row i and column j, where $1 \leq i, j \leq N$ (for a standard 9x9 Sudoku, $N = 9$). The possible values for each variable are from the set $D = \{1, 2, \ldots, N\}$.

The solution to the Sudoku puzzle must satisfy the following three constraints:

1. **Row Constraint:** Each row must contain all the numbers from the set D, with no repetitions.

2. **Column Constraint:** Each column must contain all the numbers from the set D, with no repetitions.

3. **Box/Sub grid Constraint:** The $N \times N$ grid is divided into $\sqrt{N} \times \sqrt{N}$ sub grids (blocks),

each of size $\sqrt{N} \times \sqrt{N}$. Each sub grid must contain all the numbers from the set D, with no repetitions.

If we extend this problem to three dimensional space (3D Sudoku), an additional dimension k (depth) is introduced, represented as $X_{i,j,k}$ where $1 \leq i, j, k \leq N$. According to Jana et al., the 3D Sudoku cube is divided into $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$ sub cubes, as shown in Figure 2.1[16]. The constraints for 3D Sudoku include:

1. **Axial and Planar Constraints:** Each plane slice in the XY, YZ, and XZ directions must form a valid 2D Sudoku. Plus, along every line parallel to the X, Y, or Z axes, the numbers $\{1, \ldots, N\}$ must be unique.

2. **Subcube Frequency Constraint:** Each sub cube of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$ contains $\sqrt{N} \times \sqrt{N}$ 2D minigrids of size $\sqrt{N} \times \sqrt{N}$ in all XY, YZ, and XZ directions. Refer to Figure 2.2.

This definition differs significantly from the simple uniqueness constraint in 2D Sudoku. In 2D Sudoku, the numbers within a sub grid are unique where each number appears exactly once. However, in the sub cubes of 3D Sudoku, the frequency of numbers becomes $\sqrt{N}$.

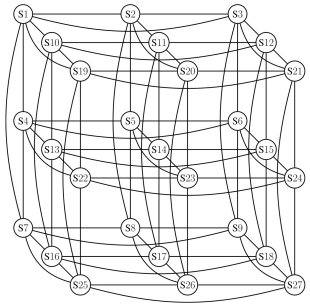


Figure 2.1: 3D Sudoku connectivity graph of 27 sub cubes (S1–S27).

2.1.2 Computational Complexity and NP-Completeness

In computational complexity theory, Yato and Seta [7] proved that the generalized Sudoku problem for $N \times N$ grids is NP-complete in 2003. Recently, Fakotakis [21] proposed a recursive Sudoku cube framework claiming polynomial-time generation via constraint propagation. This

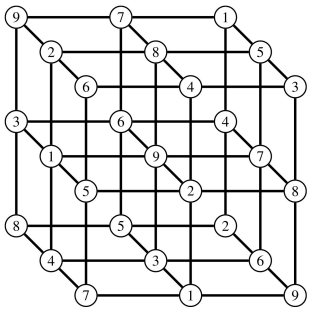


Figure 2.2: All elements inside sub cube S1.

means that as the grid size N increases, the time required to find a solution may grow exponentially. For smaller Sudoku problems (such as 9×9), traditional exact methods like backtracking or the Dancing Links algorithm (DLX), which is based on the Exact Cover problem, can find a solution in a very short time [22]. However, as the problem size increases (e.g., 16×16, 25×25), or when the problem extends to 3D (e.g., a $9 \times 9 \times 9$ cube), the size of the search space experiences combinatorial explosion.

The difficulty of solving Sudoku often stems from the puzzle configurations. As Norvig [23] noted, some randomly generated or specially designed puzzles can still trap the algorithm in deep recursion trees, especially without effective variable ordering heuristics. As the problem extends from 2D to 3D, the enlarged search space can increase the difficulty of solving complex instances, especially when relying only on deterministic search. Based on this limitation, this study investigates a hybrid framework that integrates CP with metaheuristic algorithm.

2.2 Constraint Propagation for Sudoku

Sudoku can be naturally framed as a CSP. In this framework, each cell is treated as a variable, with the domain $D = \{1, \ldots, N\}$ representing the possible numbers that can go in that cell. The rules of Sudoku, no repeated numbers in any row, column, or subgrid, are expressed as constraints [8]. CP is the process of repeatedly applying these rules to reduce the possible values for each cell, pruning the search space before or during any search for a solution. This section reviews the role of CP in Sudoku solving and explains why a propagation first step is effective in a hybrid solving framework.

2.2.1 Sudoku as a CSP and the Effectiveness of Propagation

Simonis [8] conducted one of the first thorough analyses of Sudoku as a CSP. He compared different ways to model the puzzle, various propagation algorithms, and sets of redundant constraints. Importantly, he measured how many published puzzles could be solved using only propagation, without any search. His results showed that with strong enough propagators, most standard 9×9 Sudoku puzzles could be solved entirely through propagation. This study provided strong empirical evidence that CP alone is very powerful for Sudoku.

Building on this, Norvig [23] proposed a widely cited solver that combines CP rules with a depth first search that prioritizes the most constrained cells. His experiments showed that for nearly all hard 9×9 puzzles, these propagation rules alone were enough to solve the puzzle before any search was needed. This propagation first approach directly inspires the first phase of the hybrid framework described in this thesis.

2.2.2 Constraint Propagation Techniques

CP works by iteratively narrowing down the set of valid values for each cell based on the assignments of related cells, until no further reductions can be made [24]. This process can eliminate large portions of the search space before any branching is needed.

Jana et al. [16] applied similar ideas to 3D Sudoku, using domain pruning rules to reduce candidate values before invoking backtracking search, confirming that these strategies extend naturally to higher dimensional variants.

2.3 Evolutionary and Metaheuristic Algorithms for Sudoku

Although CP can solve the majority of standard Sudoku puzzles efficiently, some more challenging instances may require additional search beyond propagation. To overcome the bot-

tlenecks of deterministic algorithms for large scale problems, researchers have attempted to introduce various metaheuristic algorithms to solve Sudoku [25, 26]. These algorithms adopt a random search strategy, which compared to the enumeration of the backtracking algorithm, strikes a balance between computational complexity and solution success rate, thereby constructing a valid solution.

2.3.1 Genetic Algorithm and Particle Swarm Optimization

GA [27, 28] is a metaheuristic search method that simulates the process of biological evolution. When applied to Sudoku solving, the most common approach is to encode the Sudoku grid as a set of chromosomes, where each gene corresponds to a value in the grid [13, 29]. The algorithm generates the next generation of candidate solutions from the current population through operators such as selection, crossover, and mutation. Each solution is evaluated based on a fitness function that measures how well it satisfies the constraints [22].

Mantere and Koljonen [13] pointed out that the main difficulty of applying GA to Sudoku lies in the fact that standard crossover and mutation operators randomly change gene values, which can easily lead to invalid grids that violate row, column, or subgrid constraints. For example, a simple single point crossover may cause duplicate numbers in the same row. To reduce the probability of generating invalid solutions, they had to design specific fitness functions to penalize conflicts or modify standard operators to maintain the legality of the permutation.

A hybrid GA proposed by Deng and Li [30] further introduced specialized permutation operators to restrict mutation to specific regions which improves the efficiency of generating valid solutions.

PSO is a swarm intelligence algorithm that simulates the social behavior of bird flocking and fish schooling. In standard PSO, a swarm of particles moves within the solution space, with each particle's movement being guided by its personal best position and the global best position [31]. However, traditional PSO is defined for continuous spaces, where position and velocity update formulas rely on vector addition and scalar multiplication, which cannot be directly applied to discrete combinatorial problems like Sudoku.

Moraglio and Togelius proposed Geometric PSO [14], a method designed for discrete

spaces like Sudoku. In Geometric PSO, a particle updates its position by blending its current solution with the best found solution (pbest and gbest), creating a new candidate solution that lies between them in the search space. This gave the particles a logical way to navigate the search space. However, despite this structured approach, experiments revealed that the swarm often lacked the diversity to solve difficult puzzles, frequently getting trapped in local optima. This observation matches Lewis' findings [22] that in highly constrained problems, metaheuristic algorithms can become trapped in local optima.

2.3.2 Other Metaheuristic Methods

In addition to GA and PSO, various metaheuristic algorithms have been explored for solving Sudoku. Lewis [22] discussed the application of Simulated Annealing (SA) in his research on metaheuristic algorithms. This method defines a cost function to evaluate the number of conflicts on the grid and introduces a temperature parameter to control the probability of accepting worse solutions, aiming to help the search process escape from local optima.

Additionally, methods like Cuckoo Search, Harmony Search, and Tabu Search have also been investigated.

While these metaheuristics offer different search strategies, Sudoku is essentially a CSP. In Sudoku, an effective way to reduce the search space is to check constraint satisfaction and prune invalid options at each step. Therefore a constructive method which builds a solution incrementally while ensuring constraint satisfaction is more suitable for Sudoku.

Based on this insight, we will focus on ACO as our metaheuristic algorithm for Sudoku. As a constructive algorithm, ACO naturally builds solutions step by step, making it highly suitable for solving Sudoku. Since the proposed method in this thesis is based on ACS, which is a specific variant within the ACO family, the following section first introduces the general ACO framework and then clarifies the relationship among ACO, AS, and ACS.

TAIPEI TECH

2.4 Ant Colony Optimization Framework

ACO is inspired by the foraging behavior of ants, as observed by Dorigo et al [32]. Ants communicate by depositing pheromones along the paths they traverse. The higher the pheromone concentration on a path, the higher the likelihood that it will be selected by following ants, thereby creating a positive feedback mechanism that leads to the convergence toward the optimal path.

2.4.1 Application of ACO to Constraint Satisfaction Problems

The ACO metaheuristic was originally designed for routing problems. However, Solnon [33] demonstrated that ACO can be effectively adapted to solve CSPs. By formulating a CSP as a graph based search, artificial ants can incrementally construct a solution by traversing the graph.

In the standard procedure of ACO, the state transition probability rule is defined as Equation 2.1 [32]:

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{il}]^{\alpha} \cdot [\eta_{il}]^{\beta}} \tag{2.1}$$

Where: - P_{ij}^{k} denotes the probability that ant k moves from state i to state j,

- i denotes the current state or node,

- j denotes one feasible next state or node,

- N_i^k denotes the set of feasible next states available to ant k at state i,

- τ_{ij} denotes the pheromone level on edge (i, j),

- η_{ij} represents the heuristic information or desirability of moving from i to j,

- α and β are control parameters that determine the relative influence of pheromone information and heuristic information.

2.4.2 From ACO to AS and ACS

ACO is a general metaheuristic framework that includes several algorithmic variants [34, 35]. Among them, AS is commonly regarded as one of the earliest recognized implementations of ACO, while ACS is a later variant proposed to improve the balance between exploration and exploitation and to accelerate convergence [32].

Compared with AS, ACS introduces three major modifications. First, ACS adopts a pseudo random proportional rule for state selection [36]. Instead of always selecting the next state purely according to the probabilistic transition rule, ACS uses a threshold parameter q_0 to switch between greedy exploitation and probabilistic exploration.
This mechanism makes ACS more aggressive in exploiting high quality choices while still preserving stochastic search capability.

Second, ACS performs local pheromone update during solution construction. In AS, pheromone is typically updated only after all ants complete their tours or constructions. In contrast, ACS updates pheromone immediately after an ant traverses an edge or makes a construction step. This local update reduces the attractiveness of the recently used component and encourages subsequent ants to explore alternative choices.

Third, ACS modifies the global pheromone update strategy. In AS, all ants may contribute to pheromone reinforcement, whereas ACS typically allows only the best ant, such as the iteration best or best so far ant, to perform the global update. This design strengthens the best solution more directly and improves convergence efficiency.

These differences are especially relevant to the present study because the proposed 3D Sudoku solver relies on iterative solution construction, local pheromone adjustment during construction, and best solution reinforcement after each iteration. Therefore, although ACO provides the general theoretical foundation, the actual search mechanism adopted in this thesis is ACS rather than the basic AS framework.

To summarize the above discussion, the main differences between AS and ACS are listed in Table 2.1.

TAIPEI TECH

Table 2.1: Comparison between Ant System (AS) and Ant Colony System (ACS)

Aspect	**Ant System (AS)**	**Ant Colony System (ACS)**
State selection	Pure probabilistic transition rule	Pseudo random proportional rule with q_0
Local pheromone update	Not used	Applied during construction
Global pheromone update	Usually contributed by all ants	Usually applied only by the best ant
Search tendency	More exploratory and distributed	Stronger exploitation with controlled exploration

2.4.3 Application of Ant Colony Optimization to Sudoku

Lloyd and Amos [12] proposed an ACS based Sudoku solver that integrates CP within the ant's construction process. As discussed in Section 2.2, CP is a powerful tool for pruning invalid candidates from cell domains. Unlike methods that only employ CP as a one time preprocessing step during initialization, their framework executes CP dynamically after an ant assigns a value to a cell, immediately reducing the valid candidates of related cells. This mechanism ensures that ants select only from valid value sets in subsequent steps, thereby significantly pruning the search space.

Furthermore, to prevent the algorithm from premature convergence to local optima, the study introduced a novel mechanism, Best Value Evaporation (BVE). While traditional ACS typically focus on the positive reinforcement of the global best path, the BVE mechanism introduces an evaporation factor f_{BVE} to reduce the pheromone increment $\Delta\tau_{best}$ on the current global best solution path. The update formula is defined as Equation 2.2:

$$\Delta\tau_{best} \leftarrow \Delta\tau_{best} \times (1 - f_{BVE}), \quad \text{where } 0 \leq f_{BVE} \leq 1. \tag{2.2}$$

Through this negative feedback mechanism, the system effectively diminishes the excessive influence of the global best path on subsequent ants, forcing the ant to continuously explore alternative regions of the solution space. Experimental results demonstrate that by combining real time CP with the BVE mechanism, the algorithm outperforms traditional methods such

as Backtracking, Iterated Local Search and Dancing Links. Since the proposed method in this thesis also performs solution construction under dynamic CP and pheromone based reinforcement, the ACS based Sudoku framework of Lloyd and Amos provides a suitable methodological basis. Therefore, the proposed 3D Sudoku solver in Chapter 3 adopts ACS as its core search mechanism and extends this framework to the three dimensional Sudoku setting.

2.5 Current Researches on 3D Sudoku

Compared to the wide range of research on 2D Sudoku, literature concerning 3D Sudoku remains relatively scarce. Existing studies primarily focus on the mathematical structure of 3D Sudoku [37] and on deterministic solving methods [16, 17].

In 2021, Jana et al. [17] proposed a solution method based on search trees. Their study adopted a permuted backtracking-based approach and utilized linked lists to store all possible solution paths. Although this method guarantees the discovery of all feasible solutions, the number of branches in the search tree can increase drastically when tackling complex puzzles with a high number of empty cells, leading to a significant increase in computational overhead.

To overcome the efficiency bottlenecks of pure backtracking, in 2023, Jana et al. [16] proposed an improved 3D Sudoku solver. This study introduced several functional constraints, including heuristic rules such as Naked Single, Hidden Single, and Lone Ranger, to prune the candidate value space prior to the backtracking search. Furthermore, they implemented a "bottom up" strategy, which involves solving the nine 2D planes individually before assembling them into a valid 3D solution. Although these refinements significantly improved computational efficiency, the core logic remains inherently dependent on deterministic search.

In addition, Najafian et al. [37] explored the structural properties of 3D Sudoku from an algebraic perspective, utilizing the stacking principles of Latin Squares to generate valid initial grids. While this research establishes the mathematical foundation of 3D Sudoku, it focuses primarily on problem generation rather than the efficient solving of given complex puzzles.

While traditional methods have made progress, existing 3D Sudoku solvers either rely on pure deterministic search [17] or use fixed logical rules to prune candidates before backtracking

[16]. To the best of our knowledge, although prior 3D Sudoku studies have introduced logical pruning rules before backtracking, there is still no study that separately evaluates the standalone solving ability of CP on a large public benchmark. In addition, the combination of CP with swarm intelligence algorithms has not yet been explored in the context of 3D Sudoku. This thesis addresses this gap by proposing a propagation-first hybrid solver that applies CP before invoking ACS, and by evaluating the respective roles of CP and ACS on a large public 3D Sudoku benchmark.

TAIPEI TECH

Chapter 3 Proposed Algorithm

In this chapter, we will provide a detailed introduction to the overall architecture and flowchart of the proposed algorithm, followed by an in depth analysis of its internal logic to help readers understand the operation of each step. We will also explore the hardware and software specifications related to the algorithm.

3.1 Main Method

The overall workflow of the proposed method is illustrated in Figure 3.1. The algorithm starts by reading the input 3D Sudoku puzzle and applying initial CP to reduce the search space. If the puzzle is completely solved after this stage, the solution is returned directly.

If the puzzle remains incomplete, the algorithm enters the ACS phase. In this phase, ants iteratively construct candidate solutions, while CP and local pheromone update are performed during the construction process.

After all ants complete their construction process in an iteration, the candidate solutions are evaluated. The pheromone trails are then updated globally according to the best solution found in the current iteration. To reduce the risk of premature convergence, BVE is further applied to weaken the excessive influence of the current best solution.

The above process is repeated until a valid solution is found or the termination condition is met. In this study, a timeout limit is used as the stopping criterion.

3.2 Implementation Details

In this section, we describe the implementation details of the proposed framework, including system requirements in Section 3.2.1 and the execution procedure of the algorithm in Section 3.2.2.

TAIPEI TECH

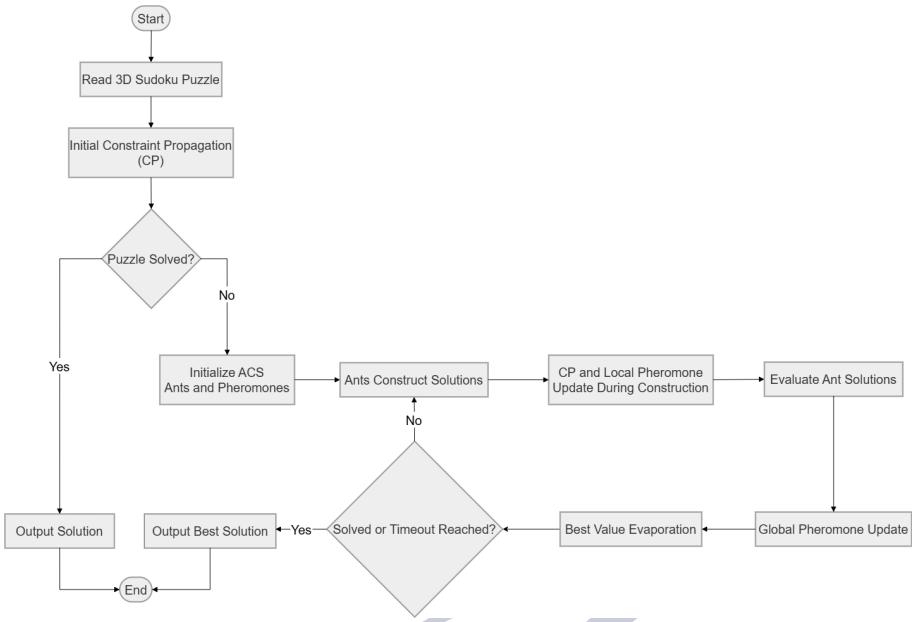


Figure 3.1: Flowchart for the Hybrid Framework of CP and Ant Colony System for 3D Sudoku

3.2.1 Requirements

As shown in Table 3.1, the system was developed using C++ version ISO C++14 and Python version 3.12.6, in a Windows 11 environment, with Microsoft Visual Studio 2022 as the primary development tool. Table 3.2 lists the Python packages used for web crawling and batch execution. The hardware configuration used in the experiments is summarized in Table 3.3. The proposed algorithm is executed on a 12th Gen Intel(R) Core(TM) i5-12500H CPU, with 8GB of memory operating at a frequency of DDR4-3200MHz and does not require GPU acceleration.

Table 3.1: Software requirements

Software Type	Software Name	Version
Operating System	Windows	11
Programming Language	C++	ISO C++14
Programming Language	Python	3.12.6
IDE	Microsoft Visual Studio Community	17.13.5
Compiler	MSVC (Microsoft Visual C++)	2022

TAIPEI TECH

Table 3.2: Python package requirements

Package Name/Version	Imported from	License
requests	Python Package Index	Apache 2.0
beautifulsoup4	Python Package Index	MIT
openpyxl	Python Package Index	MIT
subprocess	Python Standard Lib	PSF
os	Python Standard Lib	PSF

Table 3.3: Hardware requirements

Hardware Type	Name
CPU	12th Gen Intel(R) Core(TM) i5-12500H
Memory	DDR4-3200MHz 8 GB

3.2.2 Implementation Steps

According to the flowchart presented in Section 3.1, the corresponding pseudocode is outlined below. The implementation of the proposed method follows the procedure described in Algorithm 1.

Line 1 initializes the required data structures and parameters, including the number of ants, the value selection parameter q_0, the initial pheromone value τ_0, pheromone update factor ρ, evaporation parameter ϕ, and the timeout threshold. The initial pheromone value is set to $\tau_0 = \frac{1}{C}$, where C denotes the total number of cells.

Lines 2 – 9 read the input 3D Sudoku puzzle and apply CP to all pre-fixed cells. During this stage, constraint consistency is enforced, and candidate values are reduced. If the puzzle is completely solved after this step, the algorithm terminates.

If the puzzle is not solved, the ACS process is initialized (Lines 10 – 11). This includes initializing the pheromone table for each number in cells, as well as variables for storing the best solution and pheromone information.

Lines 13–16 describe the solution construction process. Each ant follows a different shuffled order of unassigned cells and incrementally assigns values to empty cells.

In Lines 17–23, the value selection strategy is performed. A random number q is generated in the range $[0, 1]$ and compared with q_0. If $q \leq q_0$, a greedy strategy is applied, where the value

Algorithm 1 ACS for 3D Sudoku

Initialize: $Ants = 81$, $q_0 = 0.85$, $\rho = 0.1$, $\tau_0 = 1/C$, $\phi = 0.001$, $T = 60$ s
Read 3D puzzle (9×9×9 grid); $C \leftarrow 729$
for each given cell i **do**
    Set cell i to its given value
    Propagate constraints across 6 unit types
end for
if all C cells are fixed **then**
    return current board ▷ Solved by CP
end if
Initialize pheromone: $\tau[i][k] \leftarrow \tau_0$ for all $i \in \{1, \ldots, C\}$, $k \in \{1, \ldots, 9\}$
bestPher $\leftarrow 0$, bestSol $\leftarrow \emptyset$
while not solved and elapsed time $< T$ **do**
    Shuffle order of unfixed cells
    for each ant a **do**
        Copy puzzle to ant's solution board
        for each unfixed cell i (in shuffled order) **do**
            $q \leftarrow \text{Uniform}(0, 1)$
            if $q \leq q_0$ **then**
                $k^* \leftarrow \arg\max_{k \in \text{candidates}(i)} \tau[i][k]$ ▷ greedy exploitation
            else
                $k^* \leftarrow$ roulette-wheel over candidates(i) with weight $\tau[i][k]$ ▷ exploration
            end if
            Set cell $i \leftarrow k^*$
            Propagate constraints across 6 unit types
            $\tau[i][k^*] \leftarrow \tau[i][k^*] \times (1 - \rho) + \rho \times \tau_0$ ▷ local pheromone update
        end for
    end for
    $Ant_{\text{Best}} \leftarrow$ ant with maximum cells filled f^*
    if $\frac{C}{C - f^*} >$ bestPher **then**
        bestSol $\leftarrow$ solution of Ant_{Best}
        bestPher $\leftarrow \frac{C}{C - f^*}$
    end if
    if $f^* = C$ **then**
        solved $\leftarrow$ true
    end if
    for each fixed cell i in bestSol **do**
        $\tau[i][k^*] \leftarrow \tau[i][k^*] \times (1 - \rho) + \rho \times$ bestPher ▷ global pheromone update
    end for
    bestPher $\leftarrow$ bestPher $\times (1 - \phi)$ ▷ Best Value Evaporation (BVE)
end while
return bestSol

TAIPEI TECH

with the highest pheromone is selected. Otherwise, a probabilistic roulette wheel selection is used, where the selection probability is proportional to pheromone. This mechanism balances exploitation and exploration.

After each assignment, CP is applied again to maintain consistency (Lines 24–25). At the same time, local pheromone update is performed using Equation 3.1:

$$\tau[i][k^*] \leftarrow \tau[i][k^*] \times (1 - \rho) + \rho \times \tau_0 \tag{3.1}$$

At the initial iteration, $\tau[i][k^*] = \tau_0$, resulting in no change after the update. This implies that all ants perform random exploration during the first iteration. After the first global pheromone update, pheromone differences begin to guide the search.

At the end of each iteration, global pheromone update is applied. The pheromone is updated based on the best performing ant, defined as the one that assigns the largest number of cells. The best pheromone value ($bestPher$) is computed as the ratio of filled cells to the total number of cells.

To prevent stagnation at local optima, BVE is applied (Lines 28 – 39). In this process, $bestPher$ is multiplied by an evaporation factor ϕ, which gradually reduces pheromone dominance and allows alternative solutions to be explored. Something worth mentioning is that when the best ant solves the puzzle, $\frac{C}{C-f^*}$ becomes 0 which would've caused Undefined Behavior (Line 29). However, in our implementation, the data type is converted to float, resulting in infinity when $f^* = C$, thus avoiding runtime errors.

Lines 13 – 39 describe one iteration of the ACO process, which is repeated until a valid solution is found or the timeout condition is reached (Line 12).

Chapter 4 Result and Analysis

4.1 Experimental Design

This section evaluates the proposed 3D Sudoku solver and analyzes the respective roles of CP and ACS under different problem settings. The default parameter settings are Ants=81, $q_0 = 0.85$, $\rho = 0.1$, $\tau_0 = 1/C$, $\phi = 0.001$, and $T = 60$ seconds.

The experiments are organized into three parts to separately assess each component of the hybrid framework. First, the solver is evaluated on the full public benchmark to measure overall effectiveness and to determine how many instances CP alone can solve. Second, a clue-density stress test from 10% to 80% investigates how the proposed CP with ACS framework behaves as the amount of initial clue information changes. Third, an additional sparse residual-search comparison from 1% to 10% clue densities compares CP with ACS with a deterministic CP with DFS baseline. This comparison is included to evaluate whether ACS or DFS is more suitable as the residual search strategy after CP.

The performance indicators used are: solved ratio (primary effectiveness measure), runtime (computational cost), and number of remaining cells after CP (residual search space size).

4.2 Public Benchmark Evaluation

4.2.1 Overall benchmark categorization

The proposed framework was evaluated on 5354 public 3D Sudoku instances, from the website [18]. Table 4.1 summarizes the results. Among all instances, 5338 (99.70%) were solved entirely by CP, while only 16 (0.30%) required ACS. CP is therefore the dominant solving mechanism on this benchmark, with ACS serving as a residual search component for a small number of difficult cases.

Table 4.1: Overall benchmark categorization

Category	Count	Percentage
CP-only solved	5338	99.70%
CP-unsolved, requiring ACS	16	0.30%
Total	5354	100%

4.2.2 Summary statistics of CP-only solved instances

Table 4.2 reports the summary statistics for the 5338 CP only solved instances.

Table 4.2: Summary statistics for CP-only solved instances

Metric	Mean	Std	Min	Max
Initial empty cells	583.31	28.875	531	637
CP runtime (ms)	4.135	1.547	1.276	8.591

The average number of initially empty cells was 583.31, ranging from 531 to 637, indicating that CP solved instances were not limited to easy or near complete puzzles. Mean CP runtime was 4.135 ms with a maximum of 8.591 ms, confirming that CP behaves efficiently and consistently on these instances.

4.2.3 Detailed results of instances requiring ACS

Table 4.3 lists the 16 instances that remained unsolved after CP, along with their initial empty cells, remaining cells after CP, CP runtime, ACS runtime, total runtime, and ACS iterations.

Most residual instances were solved in a single ACS iteration and within 50 ms. The exceptions, `sudoku3d_3494`, which required 43 iterations and 1048.010 ms and `sudoku3d_3374`, which required 9 iterations and 90.878 ms, were substantially higher than the rest. While larger residual spaces generally correspond to higher ACS cost, the relationship is not strictly monotonic. For instance, `sudoku3d_3374` retained only 216 cells after CP yet required 9 iterations and 89.414 ms, whereas `sudoku3d_1191` retained 411 cells but was solved in a single iteration within 32.171 ms. This suggests that residual search difficulty depends not only on the number of remaining cells, but also on how the remaining cells are arranged.

Table 4.3: Detailed results for the 16 ACS-required instances

puzzle_id	Initial empty cells	CP remaining empty cells	CP runtime (ms)	ACS runtime (ms)	Total runtime (ms)	Iterations
sudoku3d_223	603	216	1.603	11.789	13.392	1
sudoku3d_277	605	216	4.316	14.059	18.375	1
sudoku3d_477	614	66	5.504	8.062	13.566	1
sudoku3d_1149	637	216	4.219	17.847	22.066	1
sudoku3d_1191	629	411	3.689	32.171	35.86	1
sudoku3d_1268	633	606	2.153	46.634	48.787	1
sudoku3d_1341	631	216	4.19	19.656	23.846	1
sudoku3d_2036	625	76	4.901	8.076	12.977	1
sudoku3d_2590	623	439	3.74	45.676	49.416	1
sudoku3d_2696	633	309	4.655	46.301	50.956	1
sudoku3d_3374	601	216	1.464	89.414	90.878	9
sudoku3d_3494	605	420	2.745	1045.265	1048.01	43
sudoku3d_3778	621	152	1.909	6.634	8.543	1
sudoku3d_3830	616	216	1.448	12.447	13.895	1
sudoku3d_4381	627	216	4.649	18.992	23.641	1
sudoku3d_4518	631	306	2.275	16.553	18.828	1

4.3 Clue-Density Stress Test under 10%–80% Clue Densities

4.3.1 Motivation and dataset construction

The public benchmark reveals that CP solves nearly all instances directly, which means the benchmark alone does not expose the limits of the framework under information scarce conditions. To address this, 50 complete 3D Sudoku solutions were randomly selected, and for each, 8 instances were constructed by retaining 10%, 20%, 30%, 40%, 50%, 60%, 70%, and 80% of the cells as clues, yielding 400 generated instances in total.

4.3.2 Summary results across clue-density levels

Table 4.4 summarizes the stress test results.

Table 4.4: Clue-density stress-test generation summary

Clue density	Number of instances	CP-only solved ratio	ACS solved ratio	Overall solved ratio	Mean runtime (ms)	Mean iterations	Mean remaining cells after CP
80%	50	100%	0%	100%	4.081	0	0
70%	50	100%	0%	100%	4.404	0	0
60%	50	100%	0%	100%	4.425	0	0
50%	50	100%	0%	100%	4.463	0	0
40%	50	100%	0%	100%	4.476	0	0
30%	50	100%	0%	100%	4.569	0	0
20%	50	100%	0%	100%	4.672	0	0
10%	50	40%	52%	92%	5693.789	77.4	305.52

At densities from 20% to 80%, CP solved all 50 instances at each level, with mean runtimes

between 4.081 ms and 4.672 ms and zero remaining cells after CP. At 10% density, CP only solved ratio dropped to 40%. ACS recovered an additional 52%, raising the overall solved ratio to 92%, but at the cost of a mean runtime of 5693.789 ms, 77.4 mean iterations, and 305.52 mean remaining cells after CP. These results confirm that the proposed framework handles moderate to high clue densities efficiently, with performance degrading only when clues become extremely sparse.

4.3.3 Solved-ratio analysis

Figure 4.1 compares the CP only and overall solved ratios across clue densities. Both curves remain at 100% from 20% to 80%. At 10%, the CP only ratio drops sharply to 40%, while the overall ratio holds at 92% due to ACS. Within this dataset, ACS becomes practically necessary only at the 10% clue density level, where deterministic CP can no longer propagate most puzzles to completion.

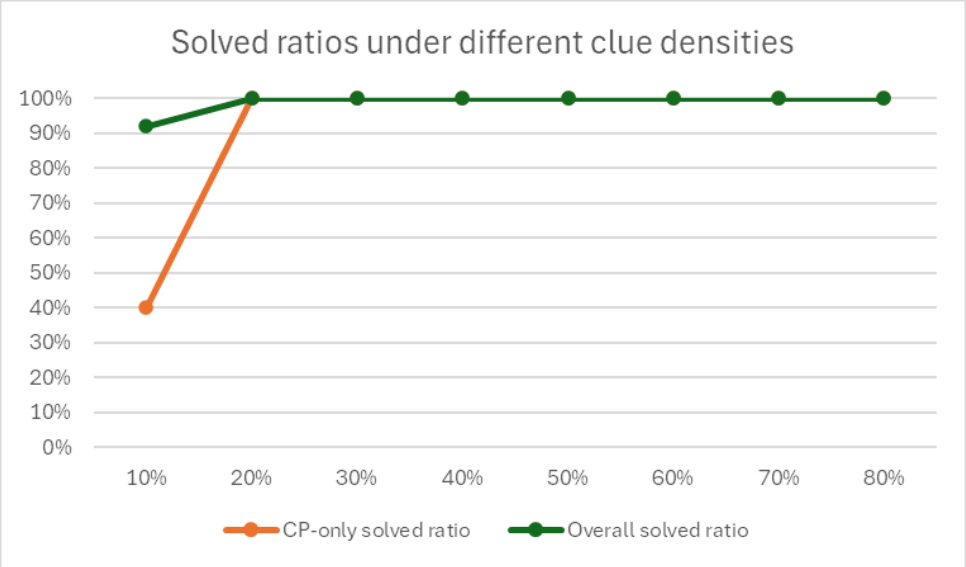


Figure 4.1: CP-only and overall solved ratios under different clue densities.

4.3.4 Runtime analysis

Figure 4.2 presents mean runtime across clue densities. Runtime stays nearly flat from 20% to 80%, then jumps to 5693.789 ms at 10%. This increase is driven by ACS replacing CP

as the dominant cost, with stochastic search operating over a large residual space.

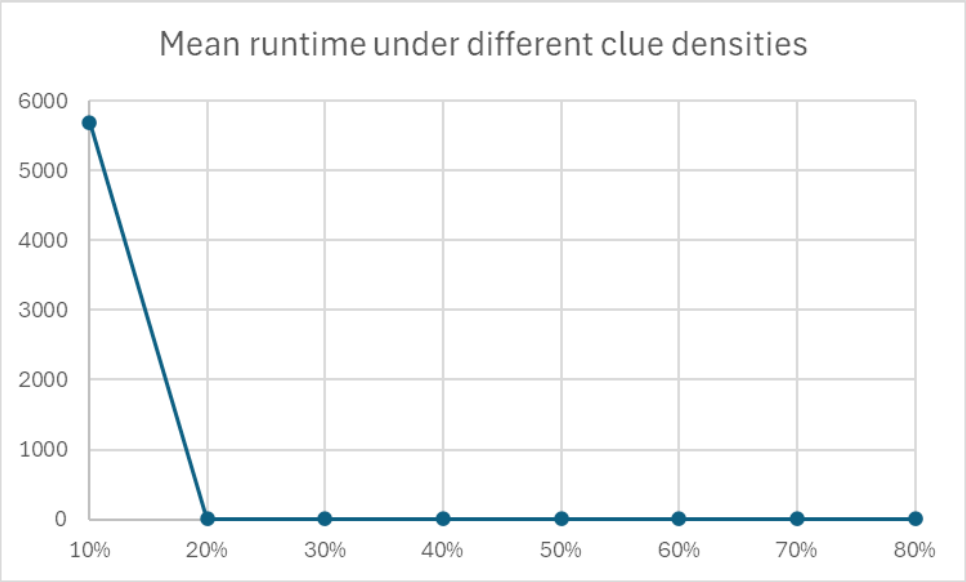


Figure 4.2: Mean runtime under different clue densities.

4.3.5 Residual-space analysis after CP

Figure 4.3 shows the mean number of remaining cells after CP. From 20% to 80%, CP leaves no residual cells, therefore ACS is never activated at these densities. At 10%, the mean rises to 305.52 cells, which explains the sharp increase in runtime and iteration count observed at this level.

Together, the three analyses above show that the proposed framework operates in two distinct regimes. At clue densities of 20% or above, CP alone completes the search with consistent sub 5 ms runtime and zero residual space. Below this threshold, CP's pruning power weakens substantially, leaving a large residual space that ACS must explore at considerably higher cost.

TAIPEI TECH

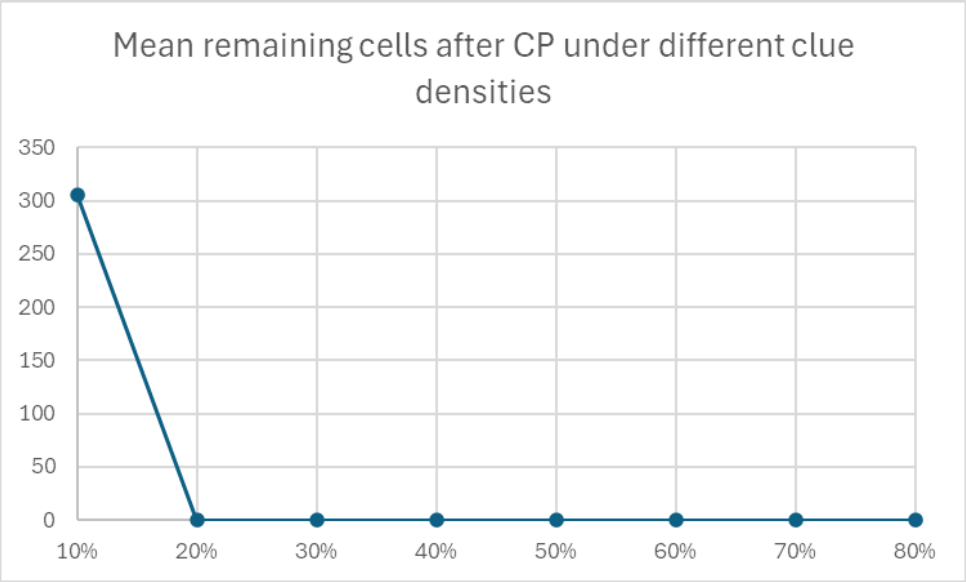


Figure 4.3: Mean number of remaining cells after CP under different clue densities.

4.4 Sparse Residual-Search Comparison between ACS-Based and DFS-Based Search

4.4.1 Motivation and experimental setup

The previous clue-density stress test showed that the proposed CP with ACS framework becomes more difficult to solve when the clue density decreases to 10%. To further examine the behavior of residual search under even sparser settings, an additional experiment was conducted using clue densities from 1% to 10%.

In this experiment, 50 solved instances were randomly selected from the public benchmark solutions. For each selected solution, 10 sparse puzzles were generated by retaining clues according to clue densities from 1% to 10%. As a result, a total of 500 sparse instances were generated. Both CP with ACS and CP with DFS were evaluated on the same generated instances to ensure a fair comparison. The timeout for this density sweep was set to 10 seconds.

The CP with DFS baseline uses the same initial CP phase as the proposed CP with ACS framework. If CP cannot complete the puzzle, DFS is applied to the remaining candidate space. Unlike ACS, DFS follows a deterministic search process under a fixed search order. In contrast,

ACS is stochastic and may construct different search trajectories across repeated runs.

The purpose of this experiment is not to replace the proposed CP with ACS framework, but to clarify the role of ACS as a residual-search strategy after CP. Since CP already reduces the search space significantly in many $9\times9\times9$ instances, it is important to compare whether the remaining search space is more suitable for deterministic DFS or stochastic ACS.

4.4.2 Solved count comparison under 1%–10% clue densities

Table 4.5 summarizes the solved counts of CP with ACS and CP with DFS under clue densities from 1% to 10%. Each clue-density level contains 50 generated instances.

Table 4.5: Solved count comparison under 1%–10% clue densities

Clue density	Number of instances	CP with ACS solved	CP with DFS solved
1%	50	0	1
2%	50	0	0
3%	50	0	0
4%	50	0	0
5%	50	0	0
6%	50	0	1
7%	50	3	8
8%	50	7	37
9%	50	33	50
10%	50	46	50

The results show that both methods struggle under extremely sparse clue densities from 1% to 6%. In this range, the amount of initial information is too limited, and the residual search space after CP remains too large for either method to solve most instances within the time limit.

Starting from 7% clue density, both methods begin to solve more instances. However, CP with DFS improves more rapidly than CP with ACS. At 8% clue density, CP with DFS solved 37 out of 50 instances, whereas CP with ACS solved only 7. At 9% and 10% clue densities, CP with DFS solved all 50 instances, while CP with ACS solved 33 and 46 instances, respectively.

These results indicate that CP with DFS is a stronger residual-search baseline for current

9×9×9 sparse instances. Once CP reduces the candidate space sufficiently, DFS can directly exploit the remaining deterministic structure. In contrast, ACS requires repeated solution construction and pheromone updates, which introduces additional overhead and makes it less efficient on average in this setting.

4.4.3 Behavioral comparison of ACS-based and DFS-based residual search

The comparison suggests that the effectiveness of the residual-search strategy depends heavily on the size and structure of the remaining search space after CP. When the clue density is extremely low, such as 1% to 6%, CP cannot reduce the candidate space enough, and both CP with ACS and CP with DFS have difficulty finding a solution within the time limit.

When the clue density increases to 8% or higher, the remaining search space becomes more manageable. In this situation, DFS benefits more directly from the reduced branching factor. Since DFS searches the remaining candidate space deterministically, it can find one valid solution quickly when the CP-reduced structure is favorable.

By contrast, ACS relies on stochastic solution construction. Although this provides search diversity, it also introduces overhead from ant construction, probabilistic value selection, local pheromone updates, global pheromone updates, and repeated iterations. Therefore, when the residual search space is already small enough for DFS to handle efficiently, ACS may become slower and less stable than DFS.

This finding refines the role of ACS in the proposed framework. ACS should not be interpreted as a generally faster replacement for DFS on current 9×9×9 3D Sudoku instances. Instead, it should be regarded as a stochastic residual-search component whose value may appear only in specific difficult cases where deterministic search is unfavorable.

TAIPEI TECH

4.4.4 A rare sparse case where ACS succeeds when DFS fails

Although CP with DFS outperformed CP with ACS on most generated sparse instances, one rare case showed the opposite behavior. Table 4.6 presents the result for `sudoku3d_2847_d07_13`, a 7% clue-density instance.

Table 4.6: A rare sparse case where ACS succeeds when DFS fails

Instance	CP with ACS	CP with DFS
`sudoku3d_2847_d07_13`	6 / 10 runs solved	Failed within 60s
Fastest successful runtime	13.32s	–
Average successful runtime	35.85s	–

For this instance, CP with DFS failed to find a solution within 60 seconds. In contrast, CP with ACS solved the same instance in 6 out of 10 repeated runs under the same 60-second timeout. The fastest successful ACS run took 13.32 seconds, and the average runtime among successful runs was 35.85 seconds.

This case should not be interpreted as evidence that CP with ACS outperforms CP with DFS overall. The full density comparison in Table 4.5 clearly shows that CP with DFS is stronger on most sparse generated instances. Rather, this rare case suggests that the stochastic nature of ACS may occasionally provide alternative search trajectories that help when a fixed deterministic DFS ordering fails within the time limit.

Therefore, the additional comparison leads to a more precise interpretation of the proposed method. For current $9\times9\times9$ 3D Sudoku instances, CP with DFS is generally the stronger residual-search baseline. However, CP with ACS may still provide limited complementary value in rare sparse hard cases due to its stochastic search behavior.

It should be noted that the 60-second repeated-run analysis was conducted only on this rare sparse instance. The full density comparison in this study used a 10-second timeout to evaluate the overall trend across all 500 sparse instances. Under this timeout setting, CP with ACS performed worse than CP with DFS on most sparse instances. However, since ACS is a stochastic

method, its search behavior may be affected by both the timeout length and the number of repeated runs. A longer timeout or more repeated runs may provide more opportunities for ACS to explore diverse search trajectories. Therefore, a complete evaluation of ACS under sparse settings should further investigate different timeout lengths and different numbers of repeated runs. Whether more ACS-favorable cases exist under longer time limits remains an open issue for future investigation.

TAIPEI TECH

Chapter 5 Conclusion and Future Work

5.1 Conclusion

This thesis proposed a hybrid solver for three dimensional Sudoku that applies CP first and invokes ACS only when the puzzle cannot be completed by CP alone. The study was motivated by a fundamental observation about 3D Sudoku, although the three dimensional extension dramatically enlarges the search space compared to standard 2D Sudoku, it simultaneously introduces a far greater number of constraints across six unit types. This increase in constraint density gives CP a stronger pruning effect, and the experimental results confirm this insight. On a public benchmark of 5,354 instances, CP alone solved 99.70% of the puzzles, with a mean runtime of 4.135 ms. This finding suggests that a higher dimensional Sudoku structure does not necessarily imply greater difficulty for constraint based methods. Rather, the density of constraints can compensate for the growth of the search space, and CP should be regarded as the primary solving mechanism rather than a simple preprocessing step.

For the instances that CP could not fully solve, ACS was able to complete the remaining residual search in the public benchmark. However, the additional comparison with CP with DFS shows that ACS is not the most efficient residual-search strategy for current 9×9×9 sparse instances. In the 1%−10% sparse density experiment, CP with DFS outperformed CP with ACS on most generated instances, especially at 8%−10% clue densities. This suggests that once CP has reduced the search space sufficiently, deterministic DFS can exploit the remaining candidate structure more efficiently than ACS.

Nevertheless, ACS still provides stochastic search diversity. In one rare sparse hard case, CP with DFS failed within 60 seconds, while CP with ACS solved the same instance in 6 out of 10 repeated runs. This result does not imply that ACS is generally superior to DFS. Instead, it suggests that ACS may occasionally help in rare cases where a fixed deterministic search order is unfavorable.

The clue density stress test further revealed that CP and ACS play different roles depending on the amount of initial information provided. At clue densities above 20%, CP consistently

solved all generated instances within short runtime. However, as density dropped to 10% and below, CP’ s solve rate declined sharply and residual search became more important. Taken together, the results show that CP is the dominant solving mechanism for current 9×9×9 3D Sudoku, while the role of ACS should be understood as an exploratory stochastic residual-search component rather than the default fastest solver.

5.2 Future Work

Several directions remain open for further investigation based on the findings and limitations of this study.

First, the behavior of stochastic residual search under different evaluation settings should be further investigated. The current sparse comparison mainly reflects the performance of CP with ACS under a fixed 10-second timeout, while the rare-case analysis suggests that ACS may exhibit different behavior when more search time and repeated runs are allowed. Therefore, future work should design a systematic evaluation across multiple timeout settings and repeated-run configurations. Such an analysis could help identify suitable timeout lengths and repeated-run settings for evaluating ACS under sparse conditions. This would provide a clearer understanding of whether ACS search diversity can consistently contribute under sparse settings, or whether its observed advantage remains limited to isolated cases.

Second, the ACS component in the current framework uses a fixed parameter configuration throughout all experiments. However, instances with a larger residual space after CP may benefit from a different parameter setting, such as a higher number of ants or a longer timeout, compared to instances where only a small number of cells remain. Investigating adaptive or instance specific parameter strategies for ACS could improve the framework’s robustness on harder residual problems and reduce the variance in the number of iterations required across instances.

Third, the current framework adopts ACS as the metaheuristic component. Other swarm intelligence algorithms, such as Max-Min Ant System (MMAS) or PSO adapted for discrete spaces, may offer different trade offs between exploration and exploitation in the residual search

phase. Comparing these alternatives on the same set of CP unsolved instances could provide useful insight into which search strategies are best suited for the remaining problems after CP.

Finally, under very sparse clue densities, especially at 10% and below, a portion of generated instances remained unsolved within the timeout limit. These cases represent the boundary of the current framework's capability under information-scarce settings. A closer analysis of their structural properties, such as the number of remaining cells after CP, candidate distribution, and branching structure, would help improve the solver and provide a clearer understanding of the limits of the hybrid design in sparse 3D Sudoku settings.

TAIPEI TECH

References

[1] Yurong Zhang, Ye Yao, Chia-Chen Lin, and Chin-Chen Chang. “Dual-Image Reversible Data Hiding Based on Sudoku Block Mapping and Adaptive Embedding”. In: *Journal of Internet Technology* 26.2 (2025), pp. 147–163. URL: https://jit.ndhu.edu.tw/article/view/3151.

[2] Longfei Yuan, Chia-Chen Lin, and Chin-Chen Chang. “Visual Quality-Enhanced Reversible Data Hiding Scheme Based on Interpolation and Variant Sudoku Matrices”. In: *Multimedia Tools and Applications* 84 (2024), pp. 20219–20247. DOI: 10.1007/s11042-024-19618-6.

[3] M. A. P. Manimekalai, M. Karthikeyan, I. Thusnavis Bella Mary, K. Martin Sagayam, Ahmed A. Elngar, Unai Fernandez-Gamiz, and Hatıra Günerhan. “Efficient Technique for Image Cryptography Using Sudoku Keys”. In: *Computers, Materials & Continua* 77.1 (2023). DOI: 10.32604/cmc.2023.035856.

[4] K. Deshpande, J. Girkar, and R. Mangrulkar. “Security Enhancement and Analysis of Images Using a Novel Sudoku-Based Encryption Algorithm”. In: *Journal of Information and Telecommunication* 7.3 (2023), pp. 270–303. DOI: 10.1080/24751839.2023.2183802.

[5] Ming Yao, Zhong Chen, Hongwei Deng, Ximei Wu, Tongzhe Liu, and Can Cao. “A Color Image Compression and Encryption Algorithm Combining Compressed Sensing, Sudoku Matrix, and Hyperchaotic Map”. In: *Nonlinear Dynamics* 113 (2024), pp. 2831–2865. DOI: 10.1007/s11071-024-10334-2.

[6] Doris S. F. Yu, Polly Li, Wung Wai Two, and Mary Waye. “Effects of Gamified Cognitive Training to Deter the Progression of Mild Cognitive Impairment”. In: *Innovation in Aging* 7.Supplement_1 (2023). Conference abstract, p. 645. DOI: 10.1093/geroni/igad104.2098.

[7] Takayuki Yato and Takahiro Seta. “Complexity and Completeness of Finding Another Solution and Its Application to Puzzles”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E86-A.5 (2003), pp. 1052–1060.

[8] Helmut Simonis. “Sudoku as a Constraint Problem”. In: *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*. 2005.

[9] Liam Davis and Tairan Ji. *Evaluating SAT and SMT Solvers on Large-Scale Sudoku Puzzles*. arXiv preprint. 2025. DOI: 10.48550/arXiv.2501.08569.

TAIPEI TECH

[10] Sangho Lim, Eun-Gyeol Oh, and Hongseok Yang. “Learning Symmetric Rules with SATNet”. In: *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., 2022, pp. 13251–13262. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/5642b9811a9ac5281be1cc84c275f251-Paper-Conference.pdf.

[11] Sascha Mücke. *A Simple QUBO Formulation of Sudoku*. 2024. arXiv: 2403.04816 [quant-ph]. URL: https://arxiv.org/abs/2403.04816.

[12] Huw Lloyd and Martyn Amos. “Solving Sudoku with Ant Colony Optimization”. In: *IEEE Transactions on Games* 12.3 (2020), pp. 234–241. DOI: 10.1109/TG.2019.2942773.

[13] Timo Mantere and Janne Koljonen. “Solving, Rating and Generating Sudoku Puzzles with GA”. In: *2007 IEEE Congress on Evolutionary Computation (CEC 2007)*. 2007, pp. 1382–1389. DOI: 10.1109/CEC.2007.4424632.

[14] Alberto Moraglio and Julian Togelius. “Geometric Particle Swarm Optimization for the Sudoku Puzzle”. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO ’07)*. 2007, pp. 118–125. DOI: 10.1145/1276958.1276975.

[15] Chuan Wang, Bing Sun, Ke-Jing Du, Jian-Yu Li, Zhi-Hui Zhan, Sang-Woon Jeon, Hua Wang, and Jun Zhang. “A Novel Evolutionary Algorithm with Column and Sub-Block Local Search for Sudoku Puzzles”. In: *IEEE Transactions on Games* PP (2023), pp. 1–11. DOI: 10.1109/TG.2023.3236490.

[16] Sunanda Jana, Manjarini Mallik, Abhinandan Khan, Arnab Kumar Maji, and Rajat Kumar Pal. “Design and Analysis of a Modified 3D Sudoku Solver”. In: *IEEE Access* 11 (2023), pp. 27352–27368. DOI: 10.1109/ACCESS.2023.3256420.

[17] Sunanda Jana, Manjarini Mallik, Arnab Kumar Maji, and Rajat Kumar Pal. “A Novel Search Tree-Based 3D Sudoku Solver”. In: *Proceedings of the International Conference on Computing and Communication Systems (IC3S)*. Vol. 170. Lecture Notes in Networks and Systems. 2021, pp. 709–718. DOI: 10.1007/978-981-33-4084-8_68.

[18] Menneske. *Sudoku 3D*. https://menneske.no/sudoku3d/eng/. Accessed: 2026-05-07.

[19] Maxime Mulamba, Jayanta Mandi, Ali İrfan Mahmutoğulları, and Tias Guns. “Perception-Based Constraint Solving for Sudoku Images”. In: *Constraints* 29.1 (2024), pp. 112–151. DOI: 10.1007/s10601-024-09372-9.

[20] Danielle Notice, Ahmed Kheiri, and Nicos G. Pavlidis. “The Algorithm Selection Problem for Solving Sudoku with Metaheuristics”. In: *2023 IEEE Congress on Evolutionary Computation (CEC 2023)*. IEEE, 2023. DOI: 10.1109/CEC53210.2023.10254026.

TAIPEI TECH

[21] Nikos Dimitris Fakotakis. *P vs NP: A Recursive Sudoku Cube Framework: Transcending NP-hard States into P-space via Polynomial-Time Generation*. SSRN preprint. 2026. DOI: 10.2139/ssrn.6257079.

[22] Rhydian Lewis. "Metaheuristics Can Solve Sudoku Puzzles". In: *Journal of Heuristics* 13.3 (2007), pp. 387–401. DOI: 10.1007/s10732-007-9012-8.

[23] Peter Norvig. *Solving Every Sudoku Puzzle*. https://norvig.com/sudoku.html. Accessed: 2025-12-27. 2006.

[24] Apekshya Bhattarai, Dinisha Uprety, Pooja Pathak, Safal Narshing Shrestha, Salina Nakarmi, and Sanjog Sigdel. "A Study of Sudoku Solving Algorithms: Backtracking and Heuristic". In: (2025). DOI: 10.48550/arXiv.2507.09708.

[25] Yashaswini Manyam, Srinidhi Sundaram, and Apurvan Sahay. "Sudoku Solver Algorithms: A Comparative Analysis". In: *2024 First International Conference for Women in Computing (InCoWoCo)*. 2024, pp. 1–9.

[26] Kanchan Rajwar, Kusum Deep, and Swagatam Das. "An Exhaustive Review of the Metaheuristic Algorithms for Search and Optimization: Taxonomy, Applications, and Open Challenges". In: *Artificial Intelligence Review* 56.11 (2023), pp. 13187–13257. DOI: 10.1007/s10462-023-10470-y.

[27] Iksan Bukhori, Jason Felix, and Saddam Ali. "Using Genetic Algorithm to Solve Puzzle Games: A Review". In: *Journal of Computer Networks, Architecture and High Performance Computing* 6 (2024), pp. 201–211. DOI: 10.47709/cnahpc.v6i1.3348.

[28] Rewayda Razaq Mohsin. "The Genetic Algorithm: A Study Survey". In: *Iraqi Journal of Science* 63.3 (2022). DOI: 10.24996/ijs.2022.63.3.27. URL: https://ijs.uobaghdad.edu.iq/index.php/eijs/article/view/3505.

[29] Manish S. R., Varshanth Reddy Y., Yuvraj R., and Gayathri Ramasamy. "A Comparative Analysis of Solving Sudoku Using Genetic Algorithm and DLX". In: *Proceedings of the 1st International Conference on Research and Development in Information, Communication, and Computing Technologies (ICRDICCT'25)*. SciTePress, 2025, pp. 631–637. DOI: 10.5220/0013918000004919.

[30] Xiu Qin Deng and Yong Da Li. "A Novel Hybrid Genetic Algorithm for Solving Sudoku Puzzles". In: *Optimization Letters* 7.2 (2013), pp. 241–257. DOI: 10.1007/s11590-011-0413-0.

[31] Meetu Jain, Vibha Saihjpal, Narinder Singh, and Satya Bir Singh. "An Overview of Variants and Advancements of PSO Algorithm". In: *Applied Sciences* 12.17 (2022), p. 8392. DOI: 10.3390/app12178392.

TAIPEI TECH

[32] Marco Dorigo, Mauro Birattari, and Thomas Stützle. “Ant Colony Optimization”. In: *IEEE Computational Intelligence Magazine* 1.4 (2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691.

[33] Christine Solnon. “Ants Can Solve Constraint Satisfaction Problems”. In: *IEEE Transactions on Evolutionary Computation* 6.4 (2002), pp. 347–357. DOI: 10.1109/TEVC.2002.802449.

[34] Luming Yang, Tao Jiang, and Ran Cheng. *Tensorized Ant Colony Optimization for GPU Acceleration*. 2024. arXiv: 2404.04895 [cs.NE]. URL: https://arxiv.org/abs/2404.04895.

[35] Christian Blum. “Ant Colony Optimization: A Bibliometric Review”. In: *Physics of Life Reviews* 51 (2024), pp. 87–95. DOI: 10.1016/j.plrev.2024.09.014.

[36] Gözde Karataş Baydoğmuş. “Probability Selection for Solving Sudoku with Ant Colony Optimization Algorithm”. In: *2022 1st International Conference on Information System & Information Technology (ICISIT)*. 2022, pp. 161–165.

[37] Mehrab Najafian, T. Aaron Gulliver, and Morteza Esmaeili. “Construction of Standard Solid Sudoku Cubes and 3D Sudoku Puzzles”. In: *IEEE Access* 10 (2022), pp. 30180–30188. DOI: 10.1109/ACCESS.2022.3159027.

TAIPEI TECH