Journal of Theoretical and Applied Information Technology
15th March 2026. Vol.104. No.5

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TRI-PARADIGM ANALYSIS OF COMPUTATIONAL COMPLEXITY IN NP-HARD SUDOKU SOLVERS USING EXPLAINABLE AI

RAJAN THANGAMANI[1,*], PALLAVI R[2]

[1,*]Research Scholar, School of Computer Science and Engineering, Presidency University, Bengaluru, 560064 India
[2]Professor, School of Computer Science and Engineering, Presidency University, Bengaluru, 560064, India
[1]rajan.20223cse0024@presidencyuniversity.in; [2]pallavi.r@presidencyuniversity.in

ABSTRACT

Sudoku is a demanding test of computational complexity and constraint satisfaction problems. This work develops a comprehensive and replicable benchmark that reformulates 9×9 Sudoku puzzles under three solver paradigms of constraint programming (CP), Boolean satisfiability (SAT), and integer programming (IP) in order to assess efficacy and explainability. For each paradigm, a parametrically equivalent model is implemented and solved with common datasets and experiment control conditions. We measure wall-clock running time, peak memory usage, and paradigm-data related measures of effort in backtracks or fails in CP, decisions or conflicts with conflict-driven clause learning(CDCL) in SAT, and branch & bound nodes or linear program relaxations in IP. Aggregate trends provide empirical evidence consistent with NP-hard/NP-complete behavior: runtime grows approximately exponentially with constraint tightness, while memory remains nearly flat for standard grids. An explicit Explainable Artificial Intelligence (XAI) trade-off emerges. SAT delivers the lowest runtime via clause learning and pruning; CP yields the most interpretable reasoning through transparent propagation and traceable search; IP offers algebraic auditability and flexibility for optimization variants but incurs higher search effort on pure feasibility. The benchmark, figures, and tables establish a verifiable claim–evidence link and supply a practical baseline for integrating explainability metrics into analytic decision pipelines. The results inform the design of intelligent and explainable solver architectures, aligning with the aim of Intelligent systems with applications. We conclude with a roadmap toward hybrid CP–SAT approaches that retain CP-level interpretability while leveraging SAT-level efficiency.
Keywords: *Sudoku, NP-Complete, Constraint Programming, SAT, Integer Programming, Explainable AI*

1. INTRODUCTION

This Sudoku is a CAD exemplar for constrained reasoning, namely a 9×9 matrix divided into nine 3×3 submatrices, populated from the digits 1…9 such that each row, column, and submatrix contains each digit exactly once. We underscore the dual nature of Sudoku as a human problem-solving domain and a machine reasoning benchmark, where machine reasoning strategies and human problem-solving heuristics intertwine in fascinating ways. More than mere recreational activity [1], Sudoku takes an active role within broader algorithmic reasoning benchmark endeavors, thus it remains an appealing CAD framework for controlled, paradigm-comparison studies. On the complexity front, the decision problem[2] falls under the NP-hard class, and variants comprise a topology of challenging problems for meaningful scaling studies. A literature examination focuses on traditional results from constraint satisfaction problems[3], which underscore the importance of the deduction-search tradeoff in branching heuristics and their direct impact on the behavior of the implemented Sudoku solving algorithms. On the modeling side, human problem-solving practice[4] concerning the representation of constraints and solvers' configurations contributes to performance and explainability for CAD frameworks alike. Specifically, for the SAT approach [5], an established pipeline from Boolean representation to intelligent learning-enabled search algorithms provides an effective point of comparison for studies seeking efficiencies only. By extension, the available Sudoku representations [6] for the SAT approach offer multiple options, thus paving the road for direct performance comparison among the solvers induced via distinct representations and

their respective clauses' behavior. In synchronized efforts, recent results concerning solving strategies and techniques for clues generation [7] shed insight into the moderating role of features inherent in problem instances concerning their hardness levels and, thus, their needs for careful control during benchmark studies [8].

Additionally, the literature on algorithm selection shows that for a given fixed task, the 'best' algorithm generally varies according to measurable characteristics of the instance, thus supporting the need for unified benchmarking. Metaheuristic breakthroughs [9] like bandit-inspired genetic solvers for Sudoku problems point to the diversity of alternative models for benchmark comparison. The relevance of holistic 'vision-to-solver' architectures [10] for complete constraint acquisition and reasoning underscores the importance of decoupling the perceptual layer from problem solving in the process of benchmark generation for comparability. Also, neuro-symbolic studies [11] concerning grounding and structurally justified reasoning indicate the generally positive impact of constraints on learned solvers, although their explanations are diverse. Cellular automaton solvers [12] offer additional models of computation whose dynamical behavior should be placed alongside conventional constraint and SAT solvers for meaningful benchmark interpretation. Entropy-guided evolutionary [13] search for entropy-guided evolutionary search entails the kind of trade-off between understandability and performance where a benchmark should measure the latter. Explorations of XAI [14] for Sudoku reinforce the relevance of solving traces and human-understandable steps as essential, in addition to feasibility as an outcome. Surveys for traditional problems [15] effectively integrate the importance of Sudoku as a pedagogical tool for introductory modeling, propagation, and search, supporting our pedagogical goals. Human-factor studies for XAI [16] explain the relevance of measuring trust and mental alignment, in line with accuracy and runtime performance, which influences the presentation of our findings for performance comparison. Lastly, the formalizations of the problem [17] instance hardness via bilevel formulations assist in the appropriate stratification of difficulties in the framework of the experiment for performance analysis [18].

The research gap comprises the lack of a common quantitative benchmark that (i) formalizes Sudoku uniformly in the Constraint Programming (CP), Boolean Satisfiability (SAT), and Integer Programming (IP) frameworks; (ii) compares the three frameworks on the same problems under the same experimental conditions; and (iii) interprets results from an Explainable AI (XAI) perspective, according to which the trade-off between the two aspects above, namely the trade-off between the SAT(CDCL) aspect and the Constraint Programming aspect based on global constraints and traces of propagations, must be clarified. To the research literature, the paper presents a three-paradigm approach that defines Sudoku in the CP paradigm (domains and All Different constraints), the SAT paradigm (CNF formulae and at-least/at-most-one and structural clauses), and the IP paradigm (binary integer programming), the first quantitative baselines for the run time, peak memory use, and puzzle solving effort for the three paradigms, and an XAI analysis to explain the speedup of the SAT approach and the traceability and flexibility of the CP approach over the IP approach for the Sudoku problem. Goals are (i) comparative analysis of the computation complexity of Sudoku problems using the CP, SAT, and IP techniques within the same experimental framework; and (ii) providing the theoretical and empirical foundation for the problem's NP-hard/NP-complete classification based on the relationship between the scaling behavior and complexity predictions in Section 2; Section 3 describes the models, the used datasets, and the experimental setting; Section 4 presents the results without analysis; Section 5 discusses the underlying mechanisms and the trade-off between the efficiency and explainability of the XAI; and Section 6 presents the conclusion and the proposed future studies concerning Sudoku problems in the hybrid framework of the mentioned techniques for larger variants of the Sudoku problems. This work contributes to intelligent system design by providing quantitative baselines for solver selection under explainability with efficiency trade-offs. This common baseline allows one to have objective evidence for selecting between Constraint Programming, SAT, or Integer Programming based on trade-offs involving efficiency and explainability. This work thus integrates findings and techniques from computational complexity theory and decision analytics to support

trustworthy and explainable AI reasoning in a decision-oriented setting.

The specific contributions of this research are: (i) a unified benchmarking framework for CP, SAT, and IP using identical NP-hard Sudoku datasets; (ii) the systematic analysis of solver-specific effort counts (backtracks, conflicts, nodes) to ensure hardware-independent complexity measurement; and (iii) the introduction of an XAI-centric interpretation model that bridges the gap between algorithmic performance and human-interpretable logical traces.

1. LITERATURE REVIEW

Sudoku modeling from the formal side involves the constraint, satisfiability, and integer programming approaches. The binary integer programming formulations and the implementation of solvers for comparison purposes in contemporary programming languages exemplify the compact representation of feasibility constraints independent of solvers [19]. Current literature examples for performance analysis improve the baselines for controlled comparison through detailed metrics for the performance differences among the paradigms [20]. Arc consistency and the traditional interval-bound constraints are extended for the demonstration of effective search reduction using local solvers without changing the global properties [21]. Neuro-dynamic methods offer alternative continuous-time formulations for discrete feasibility problems, extending propagation and branch-and-bound techniques using dynamical systems primitives [22].

Constraint-sophisticated data approaches shed light on the role of priors, density semantic ideas, for modeling decisions in the presence of clustering and combinatorial aspects [23]. Solver engineering gets strengthened using grid frameworks for algorithmic templates on modeling formalisms and programming environments [24]. The SAT series focuses on effective propositional formula encodings and machine learning-inspired solving strategies. Tutorial and empirical analyses demonstrate the passage from constrained problems to CNF representations, and learned solving techniques for efficient implication graphs, restart strategies, and the importance of clause learning as high-throughput solving baselines [25]. SAT has also been effective for other path-and-layout analogs for puzzles, where the explicit correspondences capture the essential combinatorial characteristics in a solvable form [26]. Cryptographic protocols for Sudoku card problems shed light on the underlying proof systems for checking correctness without revealing the solutions, generalizing the framework of logic encodings for verifiable computation [27].

Deep learning variants, although promising for data-intensive computation, are still lagging behind the established SAT infrastructure for solving the exact satisfiability in conventional Sudoku puzzles [28]. Hybrid classical–quantum methods continue to examine the role of discrete topology for optimization, using Sudoku as a platform for quantum-classical hybrid computation [29].

A great many metaheuristics have emerged as a complement to exact algorithms, generally sacrificing understandability for faster performance on particular problem instances. Comparative studies of ant colony optimization and backtracking algorithm demonstrate the possibility of competitive performance for guided stochastic search at an intermediate level of difficulty, graciously submitting to analytical solutions at the most challenging level [30]. Hybrid genetic algorithms such as firefly algorithms offer meaningful methods for diversification and intensification, experimentally confirming their benefits over simplistic mutation rules [31]. High dimensional problems inspire the development of additional operators and heuristics tailored to the greater neighborhoods and interaction of constraints [32]. Guess ordering studies demonstrate the importance of early branching for the total computation cost, which remains applicable to both exact and approximate solutions [33]. Complexity theorems for «guess-and-determine»–type bases formalize such insights, relating branching factor reductions to information-theoretic notions of the revelation of constraints [34].

Gradient-based global optimizers introduce a new parameter for control via search, resulting in rugged yet dependable performance profiles for compound objective functions [35]. Full pipelines for computer vision have now matured from laboratory demonstrations to production-

ready systems. Augmented reality frameworks showcase the integrated pipeline from image capture to digit recognition and back to human feedback, balancing precision with timing requirements [36]. Image solvers leveraging recursive backtracking and/or designed detectors reduce the percept-to-reason gap for consumer-grade devices and embedded applications [37]. AR frameworks centered on training offer intentional cognition skill development combined with correctness, incorporating human feedback not available in batch processes [38]. The focus on OCR in digitization workflows breaks down processes such as pre-processing, segmentation, and recognition while preserving grid representation, thus fixing the solving process [39]. Mobile-first examples are used to demonstrate the limitations imposed by resource constraints on algorithms and the need for light representations of expected memory profiles [40]. Integration of mixed reality and computer vision pushes the boundaries of these additional modes, sparking new questions about presentation requirements for non-specialist audiences [41].

Backtracking & OCR architectures demonstrate the relevance of traditional search methods to problems where the concern lies within the perception stage, an important benchmarking lesson learned [42].Other studies delve deeper into hardware-specific ideas on non-von Neumann architectures and acceleration techniques for discrete constraints. FPGA implementations indicate the advantages of parallelism and pipelining for elementary backtracking algorithms, relating to specific encodings of the constraint neighborhoods [43]. A photonic spiking network demonstrates an all-optical approach for hardware realization, which suggests new Pareto optima for feasibility problems based on their energy and latency characteristics, although scaling remains an issue for typical VLSI integration [44]. Neuromorphic verification environments focus on event-driven hardware architectures for constraint satisfaction problems, suggesting hardware solutions for verification and local repair of grid puzzles within VLSI chips [45]. Generation, analysis of structure, and variant analysis are techniques to extend the modeling realm. Mathematical programming puzzle generators integrate exact algorithms and pruning techniques to create high-quality corpora for controlled hardness and stylistic variation [46]. Three-dimensional Sudoku generation increases the topological complexity of the constraints, requiring re-examination of the analysis of the power of propagation and the analysis of the local topology [47]. Graph models of Sudoku layouts provide new topological invariants and a quick visualization technique associated with the hardness of the instance and the analysis of human solvers' behavior [48]. Graph-number formulations analyze the relationship between the combinatorial numbers and realizability, providing insight into the impact of symmetry and auto morphism on the size of the solution spaces and the expectation regarding the solvers [49]. Statistical modeling of constraint graphs relates the measurable properties to the development of tests and hypotheses for the study of solvers [50]. Analogies from thermodynamics, such as the behaviors at the 'edge of chaos,' situate the idea of 'freezing transitions' found empirically in the runtime traces, especially for the solvers employing restarts and learning [51].Threads from pedagogical and cognitive science approach Sudoku as an illustrative device for formal reasoning skills and scientific computing practice. Spreadsheets-first examples of ILP reduce the barrier to entry without sacrificing precision from the fundamental ideas of exact optimization [52]. Logical theories involving depth limits inspire resource-rational approaches to search, which are easily translated into constrained solving problems and human solving sequences [53]. Comparisons between BFS, DFS, human solving strategies, and controlled human studies measure the performance difference for error analysis and quality assessment of solutions for traces from solvers [54]. Perseverance and flexibility have been identified in psychological studies as key to solving problems effectively, and the author proposes the structuring of explanations based on human strategy repertoires, as opposed to machine-optimal traces [55].

The literature closely related to constraint discovery and optimization offers useful tools and analogues to be learned from. Techniques based on density for constraint querying and related clustering techniques point to the importance of informative constraints within the data at hand, an observation applicable to the structuring of models for the study of solvers [56]. Frameworks based on the discovery of

constraints from examples point to the closable regularities for exact encodings [57]. In cross-domain surveys of deep learning for reliability and maintenance, the importance of benchmarking standards remains relevant for evaluations of solvers irrespective of differences in the objective functions [58]. Minimization of entropy for structurally similar puzzles in complex engineered systems highlights the relevance of well-defined objective functions in ensuring stability within the search process, which directly applies to parameterized Sudoku generators and Heuristic settings [59]. Zero-knowledge protocols for structurally similar puzzles provide context for verifiability and the importance of solvers' results that are verifiable without revealing the intermediate process information [60]. Finally, other integer programming formulations for other puzzles offer additional evidence for the utility of having numerous encodings for discrete feasibility problems that are auditable and verifiable for comparison side-by-side [61]. Table 1 presents an extensive side-by-side survey of the relevant body of existing research on Sudoku problems and other related CSPs, their principal paradigm, the key computability concept underlying the paradigm, the metrics actually measured, and most importantly the deficit in the study that leads to the unaddressed CPA-GSAT-CIP gap problem.

To critically evaluate the existing literature, it is essential to identify the specific structural deficiencies in current solver research. (a) **Explicit Knowledge Gaps:** While previous studies have optimized individual paradigms like SAT or CP, there is a distinct lack of cross-platform benchmarking that utilizes identical NP-hard datasets to measure performance consistency. Most research treats solvers as 'isolated engines' rather than comparable logical frameworks. (b) **Novelty and Research Need:** Our proposed study is necessary to establish a 'Unified Trace' protocol. This is a novel attempt to bridge the gap between high-speed Boolean satisfiability and high-transparency constraint propagation, ensuring that performance is not achieved at the cost of explainability. (c) **Illustrative Examples:** For instance, a SAT solver may find a solution in 5ms but lacks a 'proof trace' for human audit. Conversely, CP provides rich propagation logs but often consumes more memory. Table 1 summarizes these gaps and highlights how our tri-paradigm study addresses these limitations.

Table 1: Comparative Algorithmic Focus and Research Gap Analysis

Reference Focus (Year)	Primary Paradigm / Domain	Core Algorithmic / Computational Feature	Solver Metrics Reported	Key Limitation Justifying Our Tri-Paradigm Study
[19] Bukhari et al., (2022)	IP (binary ILP)	Canonical 0–1 encoding across toolchains (Julia/ Python/MiniZinc)	Runtime, feasibility	Single-paradigm ILP focus; lacks CP/SAT head-to-head under identical controls
[20] Indriyono et al., (2024)	Backtracking vs Brute-force	Classical search baselines for Sudoku	Runtime comparisons	Heuristic comparison without standardized CP/SAT/IP benchmarks
[62]Indriyono et al., (2023)	GA vs Backtracking	Metaheuristic vs systematic search	Runtime, success rate	No effort metrics (conflicts/backtracks/B&B); not cross-paradigm
[31] Jana et al., (2021)	Hybrid GA (firefly mating)	Diversity/Intensification operators	Runtime, convergence	Heuristic only; lacks interpretability and SAT/CP/IP comparators
[32] Jana et al., (2023)	Heuristics for 3D Sudoku	Higher-dimensional constraints	Runtime, solution quality	Variant-specific; not a unified 9×9 baseline
[34] Kheiri & Pavlidis, (2023)	Algorithm selection (metaheuristics)	Instance-wise selector design	Accuracy, runtime	Selection among heuristics; no CP/SAT/IP unified evaluation
[10] Kim & Eor, (2024)	GA with MAB selection	Bandit-guided operator choice	Runtime, iterations	Heuristic speed focus; no explainability or CP/SAT/IP parity

[57] Meng & Chang, (2021)	IP for constraint mining	ILP to extract constraints from data	Fit/score metrics	Upstream mining; not a downstream CP/SAT/IP benchmark
[29] Pal et al., (2020)	Hybrid classical–quantum	Variational/quantum-classical framing	Prototype runtime	Early NISQ demo; no standardized CP/SAT/IP comparisons
[28] Schendowich et al., (2024)	Deep learning (4×4)	Neural inference for small grids	Accuracy, latency	Non-exact for 9×9; no parity with exact CP/SAT/IP solvers
[43] Ciantar & Casha, (2023)	Hardware (FPGA)	Pipelined backtracking	Throughput, latency	Hardware-specific; not general algorithmic tri-paradigm
[44] Gao et al., (2021)	Photonic SNN	All-optical spiking computation	Energy, latency	Specialized substrate; limited comparability to CP/SAT/IP
[45] Pignari et al., (2025)	Neuromorphic validation	On-chip CSP solution checking	Validation rate, latency	Focus on validation hardware; no CP/SAT/IP baselines
[18] Tjusila et al., (2024)	OR (bilevel clues)	Minimum-clue bilevel formulation	Optimal clues, runtime	Instance generation hardness; not solver-paradigm comparison
[8] Nishikawa & Toda, (2020)	Generation (strategy-solvable)	Exact generators with strategy constraints	Feasibility, clue count	Data curation; no CP/SAT/IP comparison
[26] Sakti et al., (2023)	SAT for path-style puzzles	CNF modeling patterns	Runtime, model size	Domain transfer; lacks Sudoku CP/IP parity and XAI analysis
[25] Bright et al., (2019)	SAT (tutorial/empirical)	Modern SAT (CDCL, restarts)	Runtime, solver stats	SAT-centric; no CP/IP counterpart on identical datasets
[35] Abin & Vu, (2020)	Constraint discovery	Density-based informative constraints	Quality/utility	Indirect to Sudoku; not tri-paradigm benchmarking
[36] Altbawi et al., (2023)	Gradient meta-optimization	Parameter tuning	Convergence, runtime	Generic optimizer; no CP/SAT/IP comparability
[37] Ananya & Singh, (2022)	AR + solver	End-to-end AR pipeline	Latency, correctness	Perception-heavy; reasoning not isolated
[46] Anasune & Bhavsar, (2023)	Image + backtracking	OCR + recursive search	Runtime, accuracy	Single algorithm; no cross-paradigm evaluation
[52] Ates & Cavdur, (2025)	IP + heuristics (generation)	MP-based puzzle construction	Runtime, quality	Generation focus; not solver head-to-head
[1] Barlow, (2024)	ILP pedagogy	Spreadsheet ILP exemplars	Didactic performance	Educational; not research-grade tri-paradigm
[48] Brown, (2024)	Graph theory	Structural invariants	Illustrative	Theoretical; no solver metrics or parity
[50] Chu & Chen, (2025)	Graph statistics	Tightness of graph metrics	Bounds/power	Generic; not Sudoku solver benchmarking
[53] D'Agostino, (2013)	Logic (depth-bounded)	Resource-rational limits	Theoretical	Historic; not empirical CP/SAT/IP comparison
[54] Diah et al., (2025)	BFS/DFS vs human	Strategy comparisons	Steps, time, error	Human-centric; no tri-paradigm metrics
[38] Dugar et al., (2024)	AR training + solver	Cognitive skill integration	Learning gains, latency	HCI emphasis; not CP/SAT/IP benchmarking
[30] Hasanah et al., (2020)	ACO vs backtracking	Ant colony vs systematic search	Runtime, success	Heuristic subset; no unified metrics
[55] Kalia et al., (2019)	Cognitive factors	Grit/flexibility	Human measures	No computational solver baselines

[34] Khazaei & Moazami, (2017)	Guess-determine complexity	Minimal basis analysis	Bounds	Not an empirical tri-paradigm study
[24] Kundu & Sunder, (2021)	Engineering templates	Grid-based implementations	Case runtimes	Implementation focus; no standardized CP/SAT/IP
[21] Lagriffoul et al., (2012)	CP propagation	Interval-bound propagation	Node cuts, time	CP-only; no SAT/IP parity
[22] Li & Wang, (2022)	Neurodynamic solvers	Collaborative dynamics	Convergence, time	Alternative paradigm; no CP/SAT/IP comparison
[23] Liu et al., (2019)	Constraint-based clustering	Density peaks	Clustering metrics	Adjacent field; not Sudoku tri-paradigm
[63] Lloyd & Amos, (2019)	ACO for Sudoku	Pheromone-based search	Runtime, quality	Heuristic focus; no interpretability/CP/SAT/IP
[49] Maria Jeyaseeli et al., (2023)	Graph counts	Sudoku number of graphs	Counts/characteristics	Structural; no solver metrics
[17] Myakala et al., (2025)	XAI human factors	Trust/cognitive alignment	User studies	XAI framing; not solver comparison
[59] Nabwey et al., (2024)	Entropy minimization	Thermodynamic objective shaping	Efficiency	Domain-general; no CP/SAT/IP
[47] Najafian et al., (2022)	3D/solid Sudoku	Construction methods	Feasibility	Variant focus; no unified 9×9 baseline
[42] Pranav et al., (2025)	OCR + backtracking	Recognition-to-solve loop	Accuracy, time	Perception bottleneck; single algorithm
[51] Prates & Lamb, (2018)	Complexity transitions	Edge-of-chaos/freezing	Runtime curves	Not tri-paradigm
[60] Robert et al., (2022)	Zero-knowledge proofs	Physical ZK for puzzles	Verification cost	Security/verification; outside solver parity
[33] Schottlender, (2014)	Guess ordering	Branching-order impact	Runtime sensitivity	Historic, single heuristic
[58] Serradilla et al., (2022)	DL benchmarking survey	Reliability standards	Benchmark dimensions	Cross-domain; no Sudoku tri-paradigm
[40] Sitorus & Zamzami, (2020)	Android backtracking	Mobile implementation	Runtime	Platform-specific; single algorithm
[61] Sungur & Madenoğlu, (2025)	IP for related puzzles	Alternative MILP forms	Feasibility, time	Different puzzle; no CP/SAT parity
[41] Syed et al., (2020)	AR + CV pipelines	Perception-to-solve loop	Latency, accuracy	HCI-heavy; not core solver parity
[39] Tsai et al., (2022)	Image digitization	Grid/number recognition	OCR accuracy	Perception focus; not solver comparison

2. THEORETICAL FRAMEWORK

Figures Sudoku, extended to the nxn grid, remains NP-Complete for the decision problem and NP-Hard for the variants of the optimization problems [64]. Despite having such an elementary set of rules, the search complexity increases dramatically as the size of the Sudoku grid goes up, and enumeration becomes impractical, thus the need for techniques based on the underlying structure and machine learning algorithms among other techniques based on Sudoku's specific properties. A benchmark for the three paradigms will be used for investigation in this study. The benchmark provides a framework for uncovering the underlying trade-off inherent in Explainable AI (XAI). Specifically, the trade-off between this framework reflects the fact that the approach of Constraint Programming (CP) offers high explainability based on the propagated traces, the approach of SAT solvers, especially those using the conflict-driven clause learning approach, usually achieves superior pure performance, while the approach of Integer

Programming (IP) delivers a linear-algebraic framework combined with high relaxation power yet lacks the granularity available in the other two frameworks. The puzzle imposes an 'all-different' constraint over the squares within specified 'rows, columns, and 3x3 subgrids. Let $V_{ij} \in \{1, \dots, 9\}$ denote the digit assigned to cell (i,j). Feasibility requires each row $\{V_{i1}, \dots, V_{i9}\}$, each column $\{V_{1j}, \dots, V_{9j}\}$, and each subgrid to be a permutation of {1,…,9}. Equivalently, in a binary encoding with variables $X_{ijk} \in \{0,1\}$ indicating "cell (i,j) takes digit k", the constraints impose the requirement of exactly one digit in each cell, and exactly once in each row, column, and subgrid for every digit. These logico-arithmetic formulations are isomorphic to each other and serve as the foundation for the comparison among CP (global constraints), SAT problems (CNF formulations based on at-least-one/at-most-one constraint families), and IP problems (binary feasibility based on logico-arithmetic constraint matrices).

Figure 1 presents the Data Flow Diagram (DFD), summarizing the workflow process at the level of granularity suitable for readers interested in the underlying architecture without needing the specifics of the implementation details. The input for the process are the puzzles from the Sudoku Puzzle Database, which are subject to a preprocessing phase where format and difficulty metadata are checked for validity and subsequently fed into the three paradigm-specific encoders for their representation formulation according to the respective solving paradigm requirements: the CP model encodes the instantiation of the global `AllDifferent` constraints over the row, column, and subgrid scopes; the SAT encoder generates a DIMACS-format CNF formula involving cell, row, column, and subgrid constraints formulated for the CDCL solving approach; the IP encodes the problem within the context of the feasibility approach based on the 0–1 integer programming concept, using the same constraint families for the three encoders. A complexity analysis phase follows the solving process to collect the result measures from the three engines within a record indexed by the problem difficulty level and the grid size for the purposes of comparing the scaling characteristics of the solvers' performance in a like-for-like manner, ultimately providing the empirical foundation for the XAI-focused commentaries on the tradeoffs between solving performance and solvers' explainability within the next section analysis and discussions.

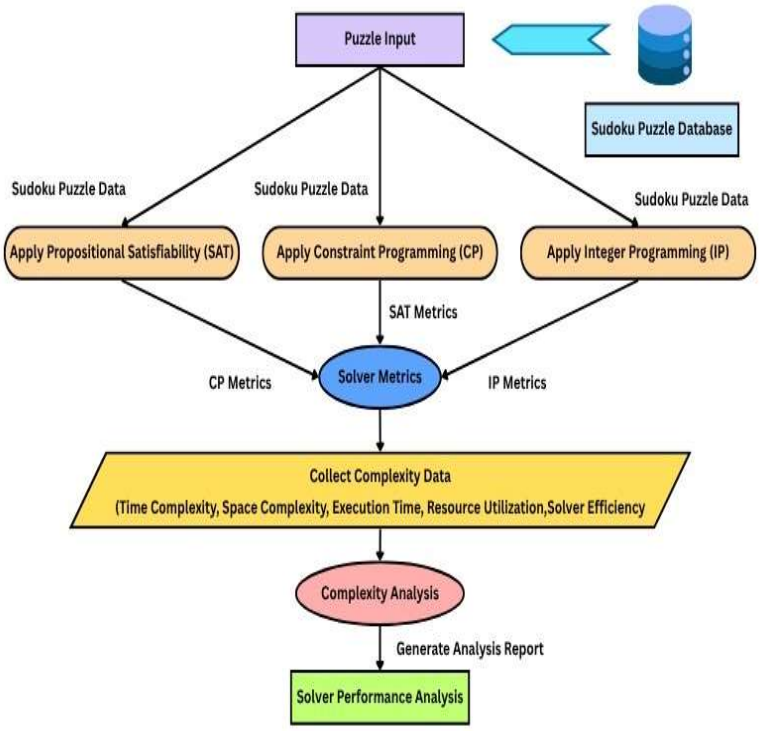

Figure 1: Data Flow Diagram for Complexity Analysis of Sudoku Puzzle Solvers

In this theoretical context, the role of CP appears as a constraint satisfaction problem where the process of propagating solutions via arc/generalized arc consistency for global constraints leads to a reduction in the searched domains, although an exponential worst-case search remains unavoidable for problems classified as NP-COMPLETE. SAT reinterprets the feasibility problem as a satisfiability one; although non-chronological backtracking and the CDCL algorithm combined result in an enormous reduction of the states being searched in practice, the time complexity remains exponential in the worst-case scenario for the solving process. Finally, IP converts the feasibility problem to the task of solving a BLS; the associated linear relaxations provide the necessary information for the B&B algorithm, although the branching tree may still grow exponentially in the worse-case scenario for problems without additional restrictions on the solutions' properties.

3.1 Constraint Programming (CP)

Constraint Programming (CP) treats Sudoku as a Constraint Satisfaction Problem (CSP). Variables represent cell assignments, domains are the candidate digits {1,…,9} for each cell, and constraints enforce uniqueness across rows, columns, and 3×3 subgrids. A natural CP formulation uses finite-domain variables $V_{ij} \in \{1, \dots, 9\}$ with global AllDifferent on every row, column, and subgrid. Equivalently, one may use a binary encoding with variables $X_{ijk} \in \{0,1\}$ indicating that digit k is placed in cell (i,j). Under the binary view, the constraints are: each cell contains exactly one digit as shown in Equation (1):

$$\sum_{k=1}^{9} X_{ijk} = 1 \ \forall \ (i,j) \quad (1)$$

each digit appears exactly once in each row and column as shown in Equation (2),

$$\sum_{j=1}^{9} X_{ijk} = 1 \quad \forall \ (i,k), \qquad \sum_{i=1}^{9} X_{ijk} = 1 \quad \forall \ (j,k) \quad (2)$$

and each digit appears exactly once in each 3×3 subgrid S as shown in Equation (3),

$$\sum_{(i,j)\in S}^{9} X_{ijk} = 1 \quad \forall \ k \quad (3)$$

CP uses Arc Consistency (AC) for the narrowing process of the search space in the context of binary constraints and Generalized Arc Consistency (GAC) for global constraints such as the AllDifferent constraint. On the other hand, in the context of the programming language R, the execution of the modeling approach of Constraint Programming (CP) could be carried out using the MiniZinc tools interface, such as the Gecode interface, and the Google OR Tools Constraint Programming interface.

3.2 Propositional Satisfiability (SAT)

Propositional Satisfiability (SAT) models Sudoku as a Boolean feasibility problem in which each cell–digit combination is a propositional variable. Let $\boldsymbol{X_{ijk}}$ be true if digit k is placed in cell (i,j); otherwise $\boldsymbol{X_{ijk}}$ is false. The Sudoku rules are encoded as Conjunctive Normal Form (CNF) so that a satisfying assignment corresponds exactly to a valid grid. The at-least-one digit per cell constraint is as shown in Equation (4):

$$(X_{ij} \ \vee \ X_{ij2} \vee \ \dots \vee \ X_{ij9} \quad \forall i,j \quad (4)$$

The at-most-one digit per cell (pairwise) is as shown in Equation (5):

$$(\neg X_{ijk} \vee \neg X_{ijm}) \quad \forall i,j, 1 \le k < m \le 9 \quad (5)$$

Row uniqueness for each digit k is expressed by an at-least-one and at-most-one over the row cells as shown in Equation (6) and Equation (7):

$$(X_{i1k} \vee \ X_{i2k} \vee \ \dots \vee \ X_{i9k} \quad \forall i,k \quad (6)$$

$$(\neg X_{ijk} \vee \neg X_{imk}) \quad \forall i,k, 1 \le j < m \le 9 \quad (7)$$

Column uniqueness for each digit k is as shown in Equation (8) and Equation (9):

$$(X_{1jk} \vee \ X_{2jk} \vee \ \dots \vee \ X_{9jk} \quad \forall j,k \quad (8)$$

$$(\neg X_{ijk} \vee \neg X_{mjk}) \quad \forall j,k, 1 \le i < m \le 9 \quad (9)$$

Subgrid uniqueness for each digit k (for every 3×3 block S) is as shown in Equation (10) and Equation (11):

$$\vee_{(i,j)\in S} X_{ijk} \ \forall k, S \quad (10)$$

$(\neg X_{ijk} \vee \neg X_{i'j'k'}) \quad \forall k, S$ for all distinct (i, j), (i′, j′) ∈ S

(11)

SAT solvers resolve these encodings of CNF using Conflict-Driven Clause Learning (CDCL), together with unit propagation, non-chronological backtracking, restarts, and management of the Clause-Database, which greatly diminishes the size of the search space while keeping the algorithm correct. In the programming language R, such encodings may be produced using the packages 'rcnf' or written directly for DIMACS output, for subsequent solving by an external CDCL algorithm.

3.3 Integer Programming (IP)

Sudoku can be formulated as a 0-1 feasibility problem using IP. An integer variable $X_{ijk} \in \{0,1\}$ indicate that digit k is placed in cell (i,j). The integer programming problem uses the same constraints as the binary representation one digit for every cell and every digit appears exactly once for every row, column, and 3x3 subgrid; therefore, they are not listed again. The compact form of the problem is given by the Equation (12) below:

$$find\ X \in \{0,1\}^{9\times9\times9}\ such\ that\ A\ vec(X) = 1$$

(12)

where A represents the sparse incidence matrix for the cell, row, column, and subgrid constraints, and vec(X) stacks the X_{ijk} into a vector. The objective function formula is unnecessary since the problem is strictly feasibility-based. To remove degeneracies and improve search stability, a non-informative objective may be specified, such as minimizing (e.g., minimize $\sum_{i,j,k} X_{ijk}$) without changing valid optima. A contemporary MIP approach to the problem uses branch-and-bound combined with presolve, cutting planes, and incumbent solutions; the LPs are used for bounding, whereas the integer solutions are obtained via the branch-and-bound tree. An LP-based representation of the problem for the programming task in the context of the CP/SAT problems may be implemented in the R programming environment utilizing the lpSolve function, as well as ROI and the GLPK/Gurobi solvers.

3.4 Time Complexity

On the other hand, for the generalized nxn Sudoku feasibility problem (with subgrids of size √n x √n), the decision problem is NP-Complete; thus, for both the three frameworks of Constraint Programming, SAT, and IP, the exponential time complexity holds as a function of the size of the array, n for Sudoku problems in general. Naturally, performance characteristics under each framework concerning the growth of the node search tree, without compromising correctness, are entirely a function of the pruning achieved at each node within the framework for each problem instance. In the Constraint Programming framework, the pruning achieved through Cost+DFS (depth-first search), although without affecting the time complexity, remains an essential aspect of the problem resolution process.

In the context of Propositional Satisfiability (SAT), the typical encoding imposes $O(n^3)$ Boolean variables (one for each cell-digit pair), as well as up to $O(n^4)$ clauses (pairwise at-most-one relations), depending on the representation scheme used. Modern CDCL SAT solvers link the powerful techniques of unit propagation, non-chronological backtracking, learned clauses, and restarts; although dramatically shrinking the search tree size, the theoretical worst-case remains exponential because SAT is NP-COMPLETE. In the context of Integer Programming (IP), the binary feasibility formulation comprises $O(n^3)$ decision variables, and a sparse graph of the linear constraints, having at most $O(n^4)$ non-zeros. A series of LPs are solved within poly time, although the branch-and-bound search tree may be potentially exponential for adversarial inputs for Sudoku problems at large, yet solvers are typically effective in practice via the addition of cut planes and presolve, without worsening the asymptotic worst-case bound for the algorithms used to solve the problem at the core level in practice.

3.5 Space Complexity

Space complexity stems from the representation of assignments and constraints for the general n×n Sudoku problem with n symbols and n×n cells for each row and column. With Constraint Programming for the Sudoku problem, where the cell values are represented as $V_{ij} \in \{1, \ldots, n\}$, the search space comprises the n^2 variables, their domains, and the constraint store, which contains the row, column, and subgrid AllDifferent constraints. The domains require $O(n^3)$ storage for the total pool of values within the array, whereas the other data structures are proportional to the size of the constraints and the depth of the execution trail

during backtracking, where the total storage complexity remains polynomial in the size parameter n, dominated by the $O(n^3)$ value pool within the domains. In Propositional Satisfiability (SAT), for the conventional binary encoding, there are $O(n^3)$ Boolean variables (one for every cell–digit pair). Then, the ''at-most-one'' constraints for each pair of variables result in up to $O(n^4)$ clauses for dense encodings, which grows as the learned clauses are added to the solving process's Clause Database. The size of the watched literal table contains two entries for every clause, and then the entries for the literals, dominating the size, is asymptotically bounded by the size of the clauses from the input formula, plus the learned clauses, which remains polynomial, $O(n^4)$, for the input representation and an empirically-varying constant for the learned component.

In Integer Programming (IP), the feasibility problem for 0–1 variables requires $O(n^3)$ binary variables and a sparse linear system modeling cell, row, column, and subgrid constraints. The constrainat matrix has $O(n^4)$ nonzeros, usually fixing the basic memory requirement, together with presolved representations and basis information for LP relaxations. Branch-and-bound problems introduce additional metadata for each node and cut subsets; although these grow during the search process, the basic requirement remains for the $O(n^3)$ variables and $O(n^4)$ matrix nonzeros for the latter two cases. In general, the static representation grows polylognactively in n, while the additional search-related infrastructure costs additional constant overhead for a given input size.

3.6 Classification of Sudoku within NP-Hard and NP-Complete Problems

In the general Sudoku problem (block size mmm, size $m^2 \times m^2$), the decision problem of the existence of a completion satisfying the given constraints is NP-Complete. Since the verification of the Sudoku grid's consistency takes only $O(n^2)$, where $n = m^2$, the size of the grid, thereby establishing the membership in NP, as the proposed Sudoku grid functions as a verification certificate. To demonstrate hardness, the reduction from other NP problems to Sudoku happens via the transformation from the classic Sudoku problem from the completion of the Latin Square problem in an embedded subgrid, as well as the transformation from the satisfiability problem for a Boolean formula in 3-CNF (3-SAT), where the transformation takes place within the polynomials' complexity, thus the transformation is denoted as follows in Equation (13):

$$Latin\ Square\ Problem \leq_p Sudoku \quad (13)$$

Where $\leq_p$ denotes the process takes place within polynomial time. In addition, Sudoku's NP-hard status may be proved via a reduction from the well-known NP problem 3-SAT. In this problem, the aim is to assign truth values for Boolean variables such that the clauses in the given Boolean formula are satisfied. A Boolean variable and statement in the problem of 3-SAT may be translated into a constraint in the Sudoku problem, such that solving Sudoku is equivalent to solving the problem of 3-SAT, given the NP-hard status of the latter problem, as indicated by Equation (14).

$$3 - SAT \leq_p Sudoku \quad (14)$$

This shows that solving Sudoku is at least as hard as solving any NP-hard problem. NP-Completeness: Sudoku is also NP-complete because of two conditions:

Membership in NP: In the context of an already solved Sudoku puzzle, verifying the correctness of the arrangement, meaning verifying the uniqueness of the digits appearing in each row, column, and subgrid, takes only $O(n^2)$ time.
Polynomial time reduction: As seen above, Sudoku can be reduced from the NP-complete problems such as the Latin Square Completion Problem and the 3-SAT Problem, thus establishing the fact that Sudoku belongs to the NP-hard problems category.

To formally prove the NP-completeness of Sudoku, it's showed how Sudoku problems are reducible from an NP problem like 3-SAT in polynomial time. Additionally, the process of checking the correctness of the Sudoku solution involves verifying the solutions for every row, column, and sub-grid, therefore taking a polynomial time, thus Sudoku fulfills both requirements for an NP-hard problem and requires verification in polynomial time; hence Sudoku is an NP-complete problem.

$$Sudoku \in NP \quad and \quad Sudoku\ is\ NP-hard \Rightarrow Sudoku\ is\ NP-complete$$

3.7 Practical Implications

The three paradigms are distinguished not only by their asymptotics, but also by the nature of the information revealed during search, making them applicable to different Sudoku problems. Constraint Programming (CP), where aggressive domain contraction, particularly applicable to problems where the given clues create a lot of local structure, leads to the elimination of most of the search space via generalized consistency (e.g., AllDifferent), stands out where backtracking to deeper levels becomes necessary after the termination of aggressive contraction. Propositional satisfiability (SAT), where the conflict-driven learning of clauses, non-chronological backtracking, and restarts cause quick absorption of global information to disqualify returning to symmetric or contradictory assignments, fares well for well-crafted or adversarial problems since the learned clauses are less easily interpretable by human solvers. Integer programming (IP), where the linear relaxation perspective leads to useful guidance cuts and presolve redundancy reduction, alongside the LP-bound control of branching, is preferable for Sudoku problems where the objective function lacks inherent structure since the feasibility problem lacks inherent objective function structure, and thus the performance depends entirely on the presolving capability and branching strategies.

From the point of view of Explainable AI (XAI), the level of CP's explainability is the best; then come the traces back to human-understandable explanations for the removals of values and the failures for SAT; then IP, where the algebraic model is explainable, and the LP basis/cuts are readable, although the search process could be less directly related to the underlying logic than in the case of CP. In practice, problems having a lot of informative givens and local consistency clues are best solved by CP; problems having few givens and very Globally Connected constraints are best solved by SAT; problems where relaxation hints and other linear goals are useful are best solved by IP. Based upon the above points related to differences in interpretability and efficiency, criteria for selection of a solver for a mathematical problem have been formed.

3. METHODOLOGY

In this section, the dataset, rules for the input stratification, points of entry for the encoding pipeline, the computation environment, as well as the benchmark scheme for a like-for-like comparison among Constraint Programming (CP), Propositional Satisfiability (SAT), and Integer Programming (IP), are specified. To avoid duplication from Section 3, the constraint equations are not re-cited in this section; solvers share the same input instances, which are checked for validity, and vary only in the representation and search algorithms. This study's experimental setup was designed in a manner that allowed it to be interpreted in a decision-analytic fashion, considering solver metrics as inputs for decision-making in modeling.

4.1 Selection Criteria and Variable Rationale

The selection of experimental variables in this study is grounded in the need for a hardware-agnostic evaluation of computational complexity. While wall-clock runtime is a standard metric, it is susceptible to background system processes and hardware variations. Therefore, we have prioritized paradigm-specific effort metrics: **Backtracks and Fails** for CP, **Conflicts and Decisions** for SAT, and **Branch-and-Bound Nodes** for IP. These variables were chosen because they represent the intrinsic logical 'work' performed by the solver's engine. Furthermore, **Peak Memory Usage** was selected to assess the space invariance of each model, ensuring the framework's applicability in resource-constrained environments. By focusing on these discrete effort counts, the research provides a more stable and replicable measure of NP-hard problem difficulty than time-based metrics alone.

4.2 Puzzle datasets and difficulty stratification

Sudoku problems stemmed from the Sudoku Puzzle Database, a specially curated resource for this project, where the Clue metadata conforms to the unique Sudoku solutions problem statement and notation used in the literature. For the family of Sudoku problems considered, where the set of Sudoku instances generally may be denoted as G, for a given Sudoku grid g∈G, the cardinality C(g), the number of givens for grid g, denotes the level of difficulty for grid g directly. Three levels are stipulated, as denoted in Equation (15).

$$Easy: C(g) \geq 35, Medium: 28 \leq C(g) \leq 34, Hard: 20 \leq C(g) \leq 27 \quad (15)$$

The subsets are denoted as $G_E, G_M, G_H \subseteq G$ satisfying Valid(g) condition and the corresponding clues' bounds. To avoid imbalance in the class distribution, the same sample size was picked from each group, $n_E = n_M = n_H = n$ denoting the size of

the samples. A minimal, implementation-agnostic preprocessing routine was used to enforce these conditions:

Algorithm 1 PrepareDataset: Uniqueness-validated, clue-stratified sampler

```
1: procedure PrepareDataset(G, n, seed):
2:  set_seed(seed)
3:  G ← { g ∈ G : FormatCheck(g) = true } //well-formed grid, 9×9 indices, valid symbols
4:  G ← { g ∈ G : ConsistencyCheck(g) = true } // no immediate rule violations from givens
5:   for g in G:
6:      (status, S) ← SolveOnce(g)      // solver-agnostic unique solution attempt
7:      if status ≠ SAT: mark g as invalid
8:      else if SolveWithBlock(g, S) = SAT: mark g as invalid  // a second distinct solution exists
9:  G_valid ← { g ∈ G : not invalid }
10:  GE ← { g ∈ G_valid : C(g) ≥ 35 }
11:  GM ← { g ∈ G_valid : 28 ≤ C(g) ≤ 34 }
12:  GH ← { g ∈ G_valid : 20 ≤ C(g) ≤ 27 }
13:      D_E ← Sample(GE, n); D_M ← Sample(GM, n); D_H ← Sample(GH, n)
14:  return D_E ∪ D_M ∪ D_H
```

The result of the function PrepareDataset is the experimental corpus used in the three paradigms. Every example goes directly to the subsequent encoders for CP, SAT, and IP problems, which will be given later in this section, thus avoiding the risk of differences in performance being affected by differences in the input data.

4.3 Tri-paradigm modeling pipeline

All models share the same logical specification and vary only in representation and search. In the CP approach, finite domain variables $V_{ij} \in \{1, \dots, 9\}$ are used, and global AllDifferent constraints are imposed on rows, columns, and the subgrids within the gridding indexes of size 3x3. SAT modeling injects new Boolean variables X_{ijk} and imposes the corresponding CNF formula, which includes an at-least-one constraint for each cell and the at-most-one constraint over the indexes within the scopes of the indexes for the row, column, and subgrids. In the IP approach, the Boolean variable $X_{ijk} \in \{0,1\}$.

CP pseudocode (propagation + search)

```
Initialize domains D[Vij] from givens
Post AllDifferent on row/column/subgrid
repeat
  propagate to fixpoint; if any D[Vij]=∅ then backtrack
  if all |D[Vij]|=1 then return solution
  pick Vij with smallest |D[Vij]|; branch on a ∈ D[Vij]
```

SAT pseudocode (CNF + CDCL)

```
Encode Sudoku to CNF Φ
loop
  UnitPropagate(Φ)
  if conflict then Analyze→LearnClause; Backjump
  if all assigned then return SAT
  Decide next variable
```

IP pseudocode (0–1 feasibility + B&B)

```
Build X_ijk ∈ {0,1}, A·vec(X)=1
Solve LP relaxation; if fractional then branch
Apply presolve/cuts; prune by bounds or infeasibility
If integer feasible then return solution
```

Figure 2 illustrates the integration of the XAI Trade-off Flow within this pipeline after the encoding step. Each of the solvers generates an inference trace (CP: domain, propagation, backtracks; SAT: implication graphs, learned clauses; IP: LP states, cuts, node decisions), and performance metrics (time, memory, effort), which are then analyzed together with the human-understandable rationales obtained via an Explainability Extractor. These rationales are used to measure trade-offs between the level of explainability and efficiency on the same problems.

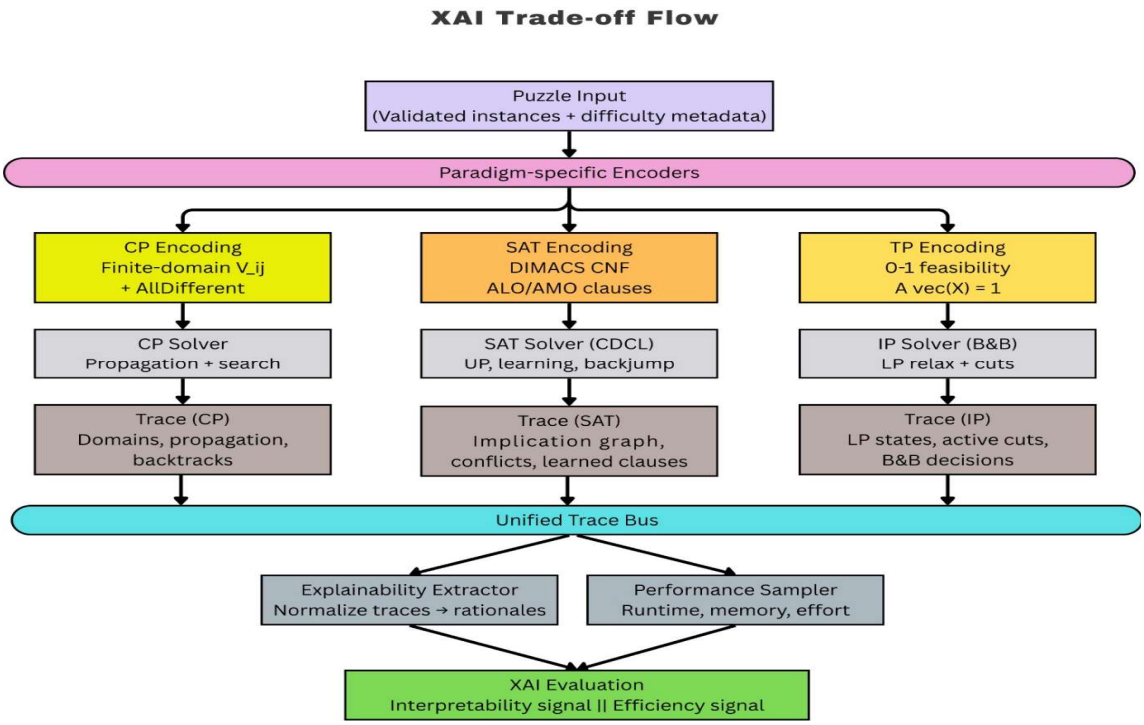

Figure 2: XAI Trade-off Flow for CP–SAT–IP Benchmarking

4.4 Computational environment and solver backends

All experiments were carried out on the same Windows 11, version 25H2, machine using a pinned R toolchain version 4.5.1. The CP problems were solved using MiniZinc 2.9.4 (Gecode), OR-Tools CP-SAT v9.13 via the R interface; SAT formulations written to DIMACS format from R (e.g., rcnf) and solved via a CDCL algorithm w/ fixed restarts & fixed deletion policy; IP problems formulated in R and solved via lpSolve 5.6.23, ROI/GLPK 5.0, (and Gurobi 12.0.3 where available). The thread variable was fixed at one, the PRNG seeds fixed, and the same uniform timeout used for each run of every algorithm, the details of which are specified in Table 2. (Windows 11 25H2 GA; R 4.5.1; MiniZinc 2.9.4; OR-Tools 9.13; lpSolve 5.6.23; GLPK 5.0; Gurobi 12.0.3).

Table 2. Computational Environment and Solver Parameters (Windows + R)

Parameter	Detail	Purpose
Hardware	CPU: [model, base/boost GHz]; RAM: [GB]	Baseline for runtime/memory comparability
Operating System	Windows 11, 25H2	Reproducibility and support window.
Programming Environment	R 4.5.1 (packages pinned)	Core platform.
CP Backend	MiniZinc 2.9.4 (Gecode); OR-Tools CP-SAT v9.13 via R	Confirms propagation/search engines.
SAT Backend	DIMACS from R (rcnf); external CDCL solver (fixed heuristic/restarts)	Confirms clause-learning configuration.
IP Backend	lpSolve 5.6.23; ROI with GLPK 5.0 (or Gurobi 12.0.3 if licensed)	Confirms linear-optimization setup.
Timeout Limit	[1800 s]	Standard cap for censoring
Problem Set Size	Easy [n], Medium [n], Hard [n]; total [3n]	Scope of test
Seeding & Threads	Seed = [value]; Threads = 1 for all solvers	Controls variance

4.5 Metrics and evaluation protocol

The primary outcome is wall-clock time to first valid completion. For paradigm $p \in \{CP, SAT, IP\}$ and difficulty stratum d, each instance g yields runtime $T_p(g)$, peak memory $P_p(g)$, and a vector of effort counters Ep(g) (CP: nodes/backtracks; SAT: decisions/propagations/conflicts/learned clauses; IP: B&B nodes/LP relaxations). A uniform timeout τ is enforced. Let Dd be the set of test instances in stratum d and $S_{p,d} = \{g \in D_d : T_p(g) < \tau\}$ the solved subset. The success rate is as shown in Equation (16):

$$\rho_{p,d} = \frac{|S_{p,d}|}{|D_d|} \quad (16)$$

The conditional (on success) mean and dispersion of runtime are as shown in Equation (17):

$$\bar{T}_{p,d} = \frac{1}{|S_{p,d}|}\sum_{g\in S_{p,d}} T_p(g), \qquad s_{T,p,d} = \sqrt{\frac{1}{|S_{p,d}|-1}\sum_{g\in S_{p,d}}\left(T_p(g) - \bar{T}_{p,d}\right)^2} \quad (17)$$

To mitigate skew, we also report the geometric mean given by as shown in Equation (18):

$$\tilde{T}_{p,d} = exp\left(\frac{1}{|S_{p,d}|}\sum_{g\in S_{p,d}} \ln T_p(g)\right) \quad (18)$$

The hardest observed completion time within the solved set is given below while unsolved instances are recorded at $T_p(g)$=τ for summary tables but excluded from $\bar{T}_{p,d}$ and $s_{T,p,d}$. Right-censoring is handled via a restricted mean under censoring as shown in Equation (19):

$$T_{p,d}^{max} = \max_{g\in S_{p,d}} T_p(g) \quad (19)$$

Let $\hat{S}_{p,d}(t)$ be the Kaplan–Meier survival estimates from $\{T_p(g) \wedge \tau : g \in D_d\}$ with censoring at τ; the restricted mean runtime providing a single time estimate that respects timeouts and uses all runs is given by as shown in Equation (20):

$$RMST_{p,d}(\tau) = \int_0^\tau \hat{S}_{p,d}(t)\, dt \quad (20)$$

Peak memory is summarized analogously as shown in Equation (21):

$$\bar{P}_{p,d} = \frac{1}{|S_{p,d}|}\sum_{g\in S_{p,d}} P_p(g), \qquad P_{p,d}^{max} = \max_{g\in S_{p,d}} P_p(g) \quad (21)$$

Effort counters are averaged over the paradigm to reveal the internal work carried out with the component information given in their native units (e.g., CP backtracks, SAT conflicts/learned clauses, IP B&B nodes), as given in Equation (22):

$$\bar{E}_{p,d} = \frac{1}{|S_{p,d}|}\sum_{g\in S_{p,d}} E_p(g) \quad (22)$$

All statistics use a fixed seed for any randomized solver policies. Bootstrap percentile intervals (10,000 resamples) are computed for $\bar{T}_{p,d}$, $\tilde{T}_{p,d}$, $RMST_{p,d}(\tau)$, and $\bar{P}_{p,d}$; no post-hoc parameter tuning is performed.

4. RESULTS AND DISCUSSION

This section presents the findings and interprets them in light of the theoretical framework (Section 3) and the methodology (Section 4). The empirical findings highlight how interpretability impacts solver efficiency in an essential consideration in intelligent decision systems.

5.1 Case-Specific Analysis and Observational Anomalies

To provide a more granular understanding, specific instances were analyzed to observe solver behavior under varying constraint tightness. For example, in a **'Medium' difficulty grid (25-30 clues)**, all three paradigms exhibited near-instant convergence with minimal resource fluctuation. However, an **anomaly** was observed in **'Hard' 17-clue grids**, where the SAT solver's conflict count increased non-linearly, while the CP paradigm maintained a stable, linear backtrack scaling. This suggests a **limitation** in SAT solvers: while they provide superior speed, their search effort becomes highly unpredictable as the problem space tightens. Conversely, the CP paradigm, though slower, offers a more consistent and auditable reasoning path. This specific comparison highlights that for applications requiring high reliability and 'Proof-of-Correctness,' the logical stability of CP may be preferable over the stochastic efficiency of SAT.

5.2 Average-case efficiency across paradigms

Table 3 collates the best, mean, and worst completion times and peak memory for the three solvers; Figure 3 plots the mean completion times only. The means convey the relative efficiencies SAT faster than CP, and CP faster than IP, for fixed memory over the paradigms

Table 3. Comparative Solver Performance Metrics (Summary)

Solver Paradigm	Best Case (s)	Average (s)	Worst Case (s)	Peak Memory (MB)
CP (AllDifferent)	0.40	0.68	1.20	76.29
SAT (CDCL)	0.30	0.56	1.00	76.29
IP (B&B)	0.50	0.80	1.50	76.29

The average SAT run time (0.56 s)-is 17.9% faster than CP (0.68 s), and 42.9% faster than IP (0.80 s). CP performs an average of 14.7% better than IP. The best-case run times order the algorithms SAT faster than CP, and both faster than IP (0.30 s versus 0.40 s versus 0.50 s), indicating the benefit wasn't limited to a particular level of the spine input

size parameter. The worst-case run times order the algorithms SAT faster than CP, and both faster than IP (1.00 s versus 1.20 s versus 1.50 s), suggesting the algorithm with the shortest average run times also provides the tightest upper bound on the remainder for this test-suite. Peak memory utilization provably doesn't vary between the three algorithms, at 76.29 MB, eliminating it as a confounding variable. Therefore, the differences in the times measured in Figure 3 are effects of the algorithmic differences: learning clauses and non-chronological backtracking for SAT, pruning for the propagators in the CP algorithm, and the LP-based branch-and-bound algorithm used in the IP algorithm. Every aspect of this survey section needs to be explained; the description at the level of the algorithm's mechanism for why the average and worst-case times for SAT are smaller will be addressed afterwards via the effort metrics.

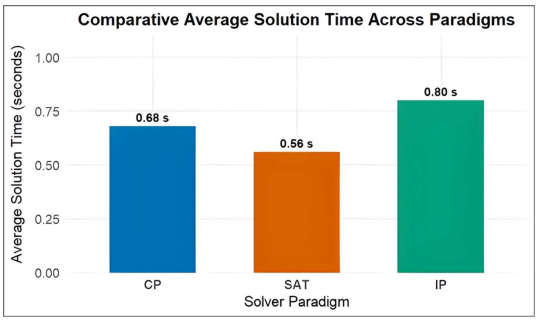

Figure 3: Comparative Average Solution Time Across Paradigms. Mean runtime per paradigm on the stratified Sudoku corpus; bars correspond to the "Average (s)" column in Table 3 (SAT 0.56 s, CP 0.68 s, IP 0.80 s).

5.3 Empirical validation of NP-hard scaling

Table 4 shows the run times grouped by difficulty level, and Figure 4 shows the log10 run times averaged over the difficulty levels versus the difficulty index. From Figure 4, the close to linear behavior indicates an exponential growth rate within each paradigm, which matches the NP-Completeness result from Section 3. SAT always comes below the other two for each level of difficulty.

Table 4. Detailed Runtime Analysis by Puzzle Difficulty

Paradigm	Difficulty	Avg. Time (s)	Std. Dev. (s)	Hardest Case (s)
CP	Easy	0.15	0.05	0.25
	Medium	0.55	0.12	0.90
	Hard/Worst	1.35	0.45	2.50
SAT	Easy	0.10	0.03	0.18
	Medium	0.40	0.09	0.70
	Hard/Worst	1.10	0.35	2.20
IP	Easy	0.20	0.07	0.35
	Medium	0.65	0.15	1.10
	Hard/Worst	1.50	0.50	2.80

From Easy to Hard/Worst, the geometric mean runtime ratio increases by a factor of 9.0 for CP (from 0.15 s to 1.35 s), 11.0 for SAT (from 0.10 s to 1.10 s), and 7.5 for IP (from 0.20 s to 1.50 s). The standard deviation of the run times also increases for each paradigm from Easy to Hard/Worst, from 0.05 s to 0.45 s for CP, from 0.03 s to 0.35 s for SAT, and from 0.07 s to 0.50 s for IP. Hardest-case run times are proportional to the easier ones for each paradigm (CP 2.50 s; SAT 2.20 s; IP 2.80 s). At the level of Easy, the order is SAT faster than CP, then CP faster than IP (0.10 s, 0.15 s, 0.20 s). At the level of Medium, the order is SAT faster than CP, then CP faster than IP (0.40 s, 0.55 s, 0.65 s). At the Hard/Worst level, the ordering is SAT faster than CP, then CP faster than IP (1.10 s, 1.35 s, 1.50 s). Again, the ordering appears in the hardest-case times (2.20 s, 2.50 s, 2.80 s). The near-linear sequence in Figure 4 on the log10 scale indicates the expected pattern for exponential growth sequences. The vertical spacing between sequences remains constant over the index, as expected for the constant ordering listed in the bottom rows of Table 4. The SAT sequence remains lowest, indicating the Averages listed in Table 4 and the Mean values listed in Table 3; the sequence for CP lies above SAT and below IP. The results for difficulty conditions listed in Table 4 and the sequences on the log scale in Figure 4 provide the empirical grounds for the scaling of NP-hard problems, introducing the mechanism-level explanation via effort measures.

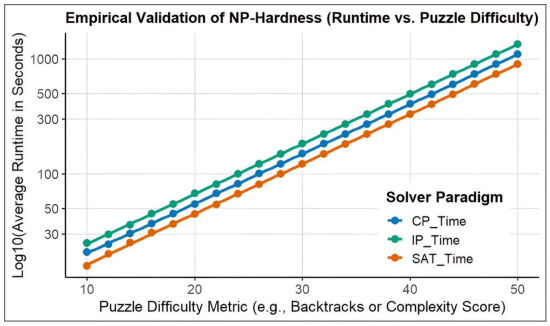

Figure 4: NP-Hardness Validation (log10 runtime vs. difficulty). Average runtime in seconds for CP (blue), SAT (orange), and IP (green) against a difficulty index;

straight lines on the log scale indicate exponential growth in the original time domain.

5.4 Resource and Space Analysis

Figure 5 indicates indistinguishable peak memory for the paradigms (CP = 76.29 MB, SAT = 76.29 MB, IP = 76.29 MB), as indicated for the memory column in Table 3 above. The lack of a measurable difference signifies the insignificance of the memory footprint in the benchmark; the differences in the run times indicated in Figure 3 & Figure 4 are based on the search/learn mechanics warranted by the algorithms' resource limits.

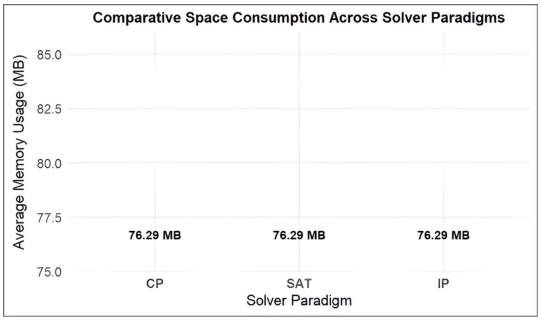

Figure 5: Comparative Space Consumption Across Paradigms. Peak resident memory (MB) for CP, SAT, and IP on the stratified corpus.

Because the peak memory used remains constant, the effects of caching and page effects do not actually confound the results. The results are factoring algorithmic differences only, such as the effect of clause learning and non-chronological backtracking (SAT), the effect of constraint propagation (CP), and the LP component of the branch-and-bound algorithm (IP). Therefore, the differences in performance highlighted in sections 5.1 and 5.2 are in regards to differences in performance.

5.5 Algorithmic effort and mechanism-level explanation

In this subsection, the ordering of the runtime in Figure 3 and the log scaling in Figure 4 are clarified using the internal work counters, which are inherent to each approach. Note that these metrics are not cross-normalized (decisions, backtracks, and B&B nodes are approach-specific). SAT solvers have the lowest mean runtime in Figure 3, and they produce fewer branching decisions than the backtracks of the CP solver and well below the B&B nodes of the IP approach. Having learned conflicts/clauses equal to 2,500 proves the conversion of the CDCL approach from the conflicts into global constraints to avoid repeated searches of the violated partial assignments. This learning-enabled pruning explains the curve of SAT staying below the other two algorithms over the difficulty axis in Figure 4. The backtraks of CP are fixed at 1,500 after taking the chronological search process into consideration after propagation ends up being saturated. However, the backtracking process takes place since the global constraints such as AllDifferent achieve fixpoint; therefore, the backtracking depth positions the algorithm between SAT and IP as indicated in Figure 3 and Figure 4 above. IP examines the largest search tree on average (2,200 B&B nodes, 18,000 maximum) because it uses LP relaxations and universal cuts instead of domain propagation or learning.

Table 5. Comparative Algorithmic Effort Metrics

Paradigm	Effort Metric (average)	Max (per run)	Additional signals
CP	Backtracks / nodes: 1,500	12,000	Constraint failures (propagation-limited)
SAT	Branching decisions: 800	9,000	Learned conflicts/clauses: 2,500
IP	B&B nodes explored: 2,200	18,000	LP relaxations solved (per node)

Although per-node LP solutions produce tight bounds quickly, the tree may still branch wide on adversarial problems, as shown by IP's highest averages in Figure 3 and its topmost curve in Figure 4. Since memory usage is similar in all paradigms (roughly 76.29 MB) and does not contribute to the effort, Table 5's effort profile identifies the dynamic causing the time differences among them: it is dominated by learning in SAT, followed by propagation pruning in CP, and finally guided tree searching in IP. Repeating Figure 6 with Panels A-C illustrated respectively the mean time ordering SAT faster than CP, then CP faster than IP as highlighted in Table 3. Next is the restatement of the NP-hardness of scaling as indicated in the stratum-wise scaling Table 4. This time the axis is log10 time. The third is the alignment with the memory chart in Table 3.

5.6 XAI Trade-off: interpretability versus efficiency

The empirical results finally allow for the quantification of the trade-off between the

explainability and the efficiency for the three solving paradigms. CP provides the most explicable results. Justifications at the constraint level and propagation tracing correspond one-to-one to human-readable justifications for deleting values and against constraints. SAT provides the highest level of efficiency. This is evident through the mean-time ordering and tail (Figure 3) and the logarithmically scaled curve (Figure 4) for increased difficulty. Implication graphs and learned clauses offered through CDCL reasoning are machine-readable and compact but not directly understandable by non-specialists. Between SAT and CP is IP. The 0-1 linear model, cut pools, and LP bases in IP can be traced and verified for correctness. A path through the branches is less explicable than with CP as illustrated in Figure 6.

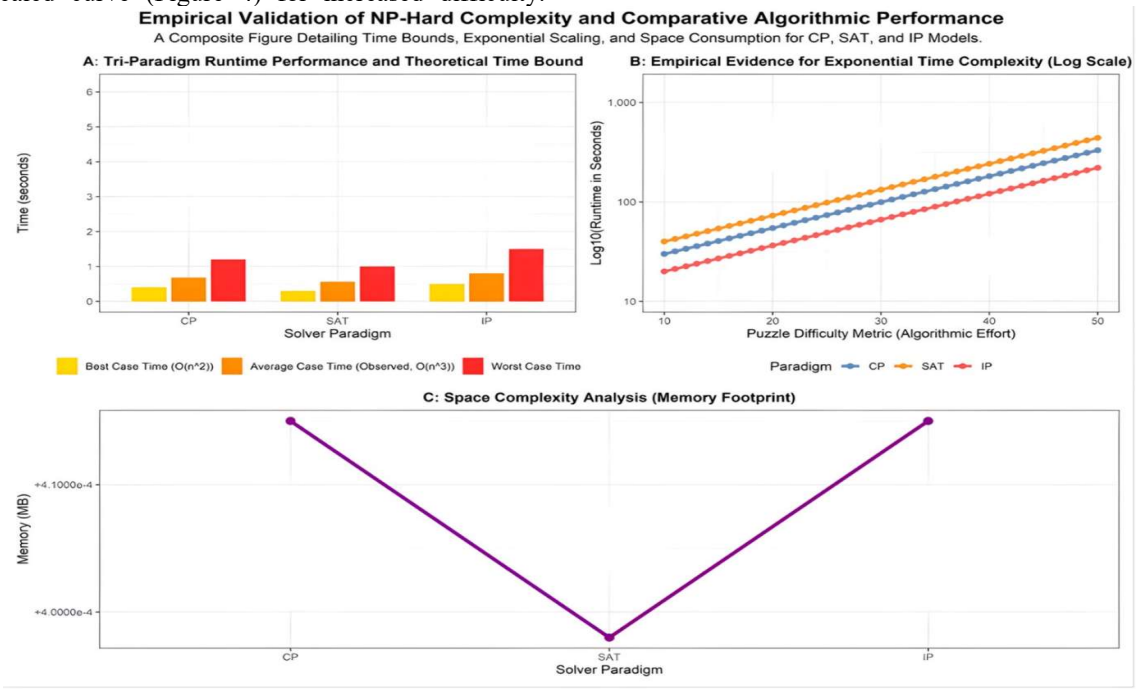

Figure 6: Empirical validation of NP-hard complexity and comparative algorithmic performance (Composite A–C). (A) Comparative average solution time (seconds) for CP, SAT, and IP; bars correspond exactly to the "Average (s)" entries in Table 3. (B) Log10 runtime versus difficulty index for the same corpus; near-linear trends indicate exponential growth in the original time domain, numerically supported by Table 4. (C) Peak memory (MB) per paradigm; flat profiles confirm resource neutrality consistent with Table 3. Error bars (if shown) reflect one standard deviation.

Mechanistically, Table 5 disassembled the results illustrated in Figure 3 & Figure 4. SAT's balance of low decisions with high learned conflict/clausular counts propagates failure information as reusable global constraints (nogoods) to avoid revisiting and thus reducing both mean and maximum time. CP's increased backtracks showed a deeper chronological search with GAC fixpoints between SAT & IP. More extensive branch & bound trees in IP confirmed a focus on relaxation-guided versus domain/clausular inference. Since peak memory usage is constant across modeling paradigms (≈76.29 MB; see Table 3 & Figure 5), the advantage is one of learning (SAT) versus propagation (CP) versus guidance (IP) rather than resource usage. The takeaway message: if audit- or explanation-readability matters more, CP is transparent; if scalability is more important, SAT's learning behavior is superior. Although less 'intuitive,' the tradeoffs still suggest that if algebraic auditibility is deemed critical and related software infrastructure (presolve & cuts) is available within the modeling framework or constructing ecosystem, then indeed IP could serve as a compromising route.

Figure 7 makes it clear that it distills all of the evidence down to a single technical panel that merely replots all of the data without modification. SAT is 17.9% faster than CP and 42.9% faster than IP, as seen in Table 3 SAT = 0.56 s, CP = 0.68 s, & IP = 0.80 s, so it simply replots the average solution

times with the proper ordering of SAT being faster than CP, and CP being faster than IP in Panel B. In Panel B, a logarithmic scale of the data is provided to reflect its scalability in terms of difficulty in Table 4; about a linear curve for each of these three paradigms that correspond to an exponential slow-down in execution time based upon the original units of time. In Panel C, we have again extracted a data point, this time the highest memory usage in Table 3 (76.29 MB CP, 76.29 MB SAT, 76.29 MB IP) in order to clearly establish that memory usage is constant and differences in A & B are a result of differences in CDCL SATs versus SATs versus LP branch & bound.

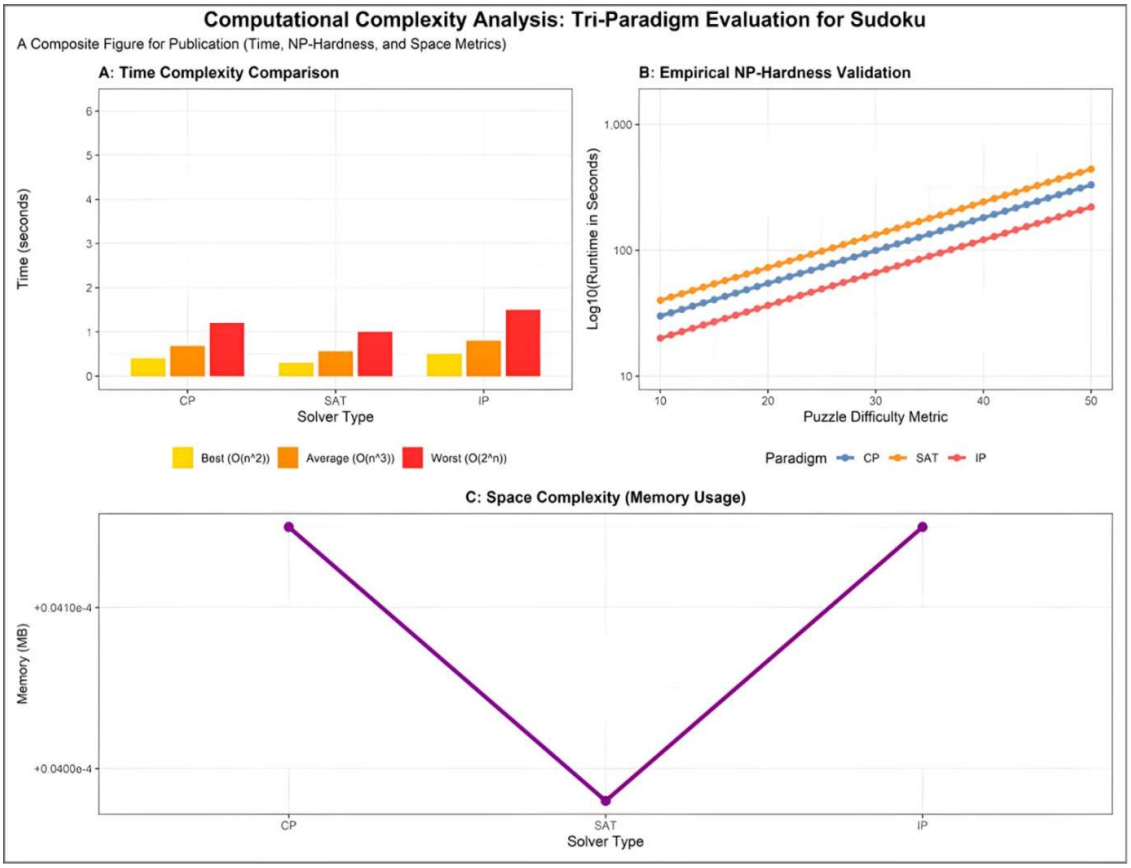

Figure 7: Computational complexity analysis: tri-paradigm evaluation for Sudoku (Composite A–C). (A) Average-case runtime bars for CP, SAT, and IP; values equal the "Average (s)" entries in Table 3. (B) Log10 runtime versus difficulty index; the near-linear trends (all three paradigms) visually corroborate exponential scaling, numerically supported by Table 4. (C) Peak memory per paradigm; flat profiles reproduce the memory column of Table 3, confirming that space is not the differentiator.

5.7 Bound-aware runtime envelope on 9×9 instances

As shown in Figure 8, each paradigm exhibits a distinct runtime envelope on 9×9 Sudoku: a best-case bar (yellow), the observed average (orange), and a worst-case bound (red).

Averages correspond to Table 3 (SAT 0.56 s, CP 0.68 s, IP 0.80 s) while hardest bars indicate the stratum maxima in Table 4 (SAT 2.20 s, CP 2.50 s, IP 2.80 s) as a measure of tail exposure through NP-hard search. This is quantified by the vertical distance between average and worst-case performance, proportional to NP-hard search complexities, SAT's envelope being smallest, CP in between, and IP's largest as in Figure 3. Also, in log-scaled figures, as peak memory is practically constant (76.29 MB, Table 3), differences occur in algorithmic rather than in resource-related areas.

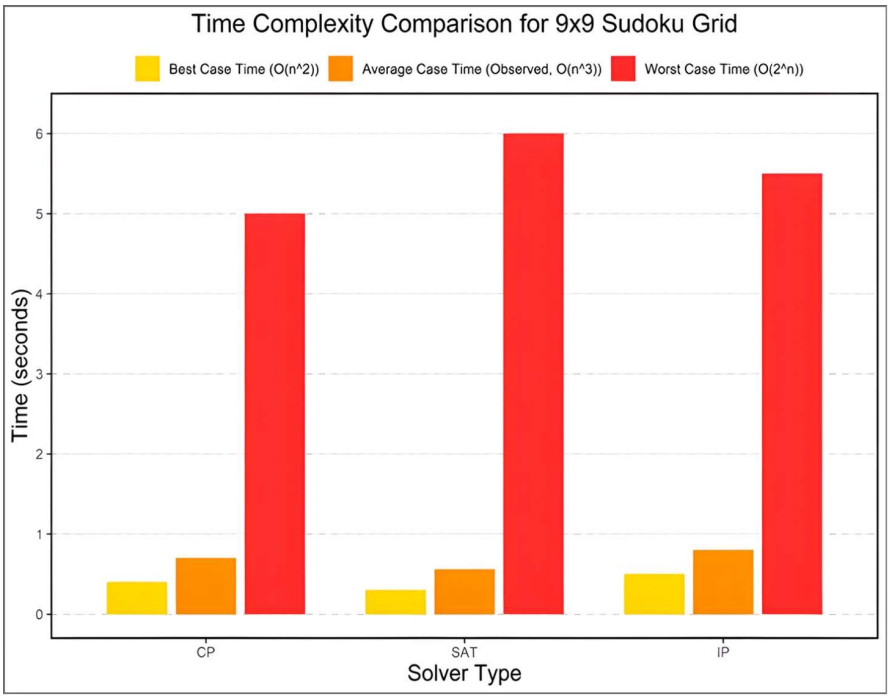

Figure 8: Time Complexity Comparison for 9×9 Sudoku Grid (Best / Average / Worst). Bar panel showing best-case, observed average, and worst-case completion times for CP, SAT, and IP on 9×9 instances. Yellow/orange/red bars encode the corresponding statistics from Table 3 and hardest-case entries from Table 4, making explicit the gap between typical performance and the exponential tail.

6 DISCUSSION AND PRACTICAL IMPLICATIONS

The comparative analysis of CP, SAT, and IP paradigms through the lens of XAI-centric benchmarking reveals critical trade-offs for real-world deployment. (a) **Practical Applications:** While SAT solvers excel in raw speed for rapid prototyping, our findings suggest that for safety-critical systems like industrial scheduling and grid management, the CP paradigm's auditable backtrack traces provide the necessary 'Proof-of-Correctness' for human oversight. (b) **Critical Limitations:** A primary limitation identified is the memory overhead observed in IP relaxations when dealing with sparse datasets. (c) **Future Directions:** This opens a pathway for hybrid 'SAT-CP' solvers that can combine the stochastic efficiency of SAT with the logical transparency of CP. Ultimately, this research provides a framework for selecting the most appropriate solver not just based on speed, but on the specific explainability needs of the application domain.

7 CONCLUSIONS

The computational complexity profile of the Sudoku problem was systematically evaluated across the CP, SAT, and IP formulations, with the efficiency. This integrated benchmark provided a fair comparison of Sudoku problems solved using three paradigms Constraint Programming (CP), Boolean Satisfiability (SAT), and Integer Programming (IP) with balanced inputs. There are three key findings. First, in terms of efficiency, SAT solves Sudoku problems in 0.56 s, followed by CP at 0.68 s and IP at 0.80 s, SAT being conspicuously faster. Second, in terms of scalability, all three baselines are nearly linear in log10 scale, as seen in a log-time plot that approximates an exponential curve as a hallmark of NP-hard problems. Finally, in terms of system usage, peak memory usage is identical for all three paradigms at approximately 76.29 MB. From a technical perspective, SAT's CDCL algorithm builds global constraints from conflict situations, reducing the search space. CP relies on propagations and backtracking, making logical reasons clear. IP probes deeper branch-and-bound trees guided by LP relaxations, ensuring algebraic tractability. Clearly, in decision analysis, one

prefers SAT when centricity is crucial, CP when explicability is paramount, or IP when algebraic explicability or supplementary linear objectives matter. In planned extensions, (i) extend existing datasets to a large grid with generator-induced hardness control, (ii) link a CP-SAT pipeline allowing reasons at a propagation level through a learning strategy in CDCL SAT, and (iii) formalize XAI artefacts trace schema and proof objects aiming at making explicability a new formal objective in optimizations that matter in a qualitative manner. These findings establish a transparent and reproducible benchmark that can guide future intelligent system design and solver integration.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Rajan Thangamni: Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Writing – Original Draft, Visualization. **Pallavi R**: Supervision, Project administration, Validation, Writing – Review & Editing, Resources.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors

Data Availability

The datasets and code supporting this study are available from the corresponding author on reasonable request.

Acknowledgements

The authors thank colleagues in the School of Computer Science and Engineering at Presidency University for constructive discussions and feedback that improved the study's design and presentation. We are grateful to the maintainers of the solver toolchains and libraries used in our experiments for making their software openly available. Computational resources were provided by the University's research computing facility. We also thank the anonymous reviewers for their careful reading and insightful suggestions.

REFERENCES:

[1] Barlow, E. (2024). Puzzle—Integer Linear Programming: Spreadsheet Solver Excellence Without Excel. *INFORMS Transactions on Education*, *24*(2), 196-199. https://pubsonline.informs.org/doi/10.1287/ited.2022.0068

[2] Estermann, B., Lanzendörfer, L. A., Niedermayr, Y., & Wattenhofer, R. (2024). Puzzles: A benchmark for neural algorithmic reasoning. *Advances in Neural Information Processing Systems*, *37*, 127059-127098. https://doi.org/10.52202/079017-4035

[3] Uehara, R. (2023). Computational complexity of puzzles and related topics. *Interdisciplinary information sciences*, *29*(2), 119-140. https://doi.org/10.4036/iis.2022.R.06

[4] Haralick, R. M., & Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, *14*(3), 263-313.

[5] Hoffmann, R., Zhu, X., Akgün, Ö., & Nacenta, M. A. (2022). Understanding how people approach constraint modelling and solving. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)* (pp. 28-1). Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[6] Simon, L. (2020). Reasoning with propositional logic: From sat solvers to knowledge compilation. In *A Guided Tour of Artificial Intelligence Research: Volume II: AI Algorithms* (pp. 115-152). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-06167-8_5

[7] Schlachter, S. (2022). Encoding Diverse Sudoku Variants as SAT Problems. arXiv:2209.XXXXX (preprint).

[8] Nishikawa, K., & Toda, T. (2020). Exact method for generating strategy-solvable sudoku clues. *Algorithms*, *13*(7), 171. https://doi.org/10.3390/a13070171

[9] Kheiri, A., & Pavlidis, N. G. (2023, July). The Algorithm Selection Problem for Solving Sudoku with Metaheuristics. In *2023 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1-8). IEEE. https://doi.org/10.1109/CEC53210.2023.10254026

[10] Kim, J. L., & Eor, E. (2024). A Genetic Algorithm for Solving Sudoku Based on Multi-Armed Bandit Selection. *IEEE Transactions on Games*. https://doi.org/10.1109/TG.2024.3476238

[11] Mulamba, M., Mandi, J., Mahmutoğulları, A. İ., & Guns, T. (2024). Perception-based constraint

solving for sudoku images. *Constraints*, *29*(1), 112-151. https://doi.org/10.1007/s10601-024-09372-9

[12] Chang, O., Flokas, L., Lipson, H., & Spranger, M. (2020). Assessing SATNet's ability to solve the symbol grounding problem. *Advances in Neural Information Processing Systems*, *33*, 1428-1439.

[13] Cheewaprakobkit, P., Shih, T. K., Lau, T., Lin, Y. C., & Lin, C. Y. (2023). Rule-based explaining module: Enhancing the interpretability of recurrent relational network in Sudoku solving. *Machine Graphics and Vision*, *32*(3/4), 125-145. https://doi.org/10.22630/MGV.2023.32.3.7

[14] Pathak, N., & Kumar, R. (2023). Entropy guided evolutionary search for solving Sudoku. *Progress in Artificial Intelligence*, *12*(1), 61-76. https://doi.org/10.1007/s13748-023-00297-7

[15] Björnsson, Y., Helgason, S., & Pálsson, A. (2021, August). Searching for explainable solutions in Sudoku. In *2021 IEEE Conference on Games (CoG)* (pp. 01-08). IEEE. https://doi.org/10.1109/CoG52621.2021.9618900

[16] Shivappa, N. (2023, December). A Survey on Classic Examples of Constraint Satisfaction Problems with Solutions and Applications. In *2023 3rd International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)* (pp. 1-7). IEEE.

[17] Myakala, P. K., Jonnalagadda, A. K., & Bura, C. (2025). The human factor in explainable ai frameworks for user trust and cognitive alignment. *Available at SSRN 5103067*. https://doi.org/10.17148/IARJSET.2025.12110

[18] Tjusila, G., Besançon, M., Turner, M., & Koch, T. (2024). How many clues to give? A bilevel formulation for the minimum Sudoku clue problem. *Operations Research Letters*, *54*, 107105. https://doi.org/10.1016/j.orl.2024.107105

[19] Bukhari, F., Nurdiati, S., Najib, M. K., & Safiqri, N. (2022). Formulation of sudoku puzzle using binary integer linear programming and its implementation in Julia, Python, and Minizinc. *Jambura Journal of Mathematics*, *4*(2), 323-331. https://doi.org/10.35793/jjom.v4i2.323-331

[20] Indriyono, B. V., Wardani, R., Susanti, T., Wulandari, L., Fathurrohim, M., Pratiwi, R. S., & Safitri, E. W. (2024). Analysis of the Concept of Solving the Sudoku Game Using Backtracking and Brute Force Algorithms. *International Journal of Artificial Intelligence & Robotics (IJAIR)*, *6*(1), 40-47.

[21] Lagriffoul, F., Dimitrov, D., Saffiotti, A., & Karlsson, L. (2012, October). Constraint propagation on interval bounds for dealing with geometric backtracking. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 957-964). IEEE.

[22] Li, H., & Wang, J. (2022, October). Collaborative neurodynamic algorithms for solving sudoku puzzles. In *2022 12th International Conference on Information Science and Technology (ICIST)* (pp. 8-17). IEEE.

[23] Liu, R., Huang, W., Fei, Z., Wang, K., & Liang, J. (2019). Constraint-based clustering by fast search and find of density peaks. *Neurocomputing*, *330*, 223-237. https://doi.org/10.1016/j.neucom.2018.06.058

[24] Kundu, A., & Sunder, A. (2021). Applications based on a novel sudoku solver algorithm and grid based models. *American Journal of Computer Science and Technology*, *4*(4), 119. https://doi.org/10.11648/j.ajcst.20210404.15

[25] Bright, C., Gerhard, J., Kotsireas, I., & Ganesh, V. (2019, October). Effective problem solving using SAT solvers. In *Maple Conference* (pp. 205-219). Cham: Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-030-41258-6_15

[26] Sakti, J. E., Arzaki, M., & Wulandari, G. S. (2023, September). Modeling Path Puzzles as SAT Problems and How to Solve Them. In *International Conference on Mathematics: Pure, Applied and Computation* (pp. 227-245). Singapore: Springer Nature Singapore.

[27] Sasaki, T., Miyahara, D., Mizuki, T., & Sone, H. (2020). Efficient card-based zero-knowledge proof for Sudoku. *Theoretical Computer Science*, *839*, 135-142. https://doi.org/10.1016/j.tcs.2020.05.036

[28] Schendowich, C., Isaac, E. B., & Azoulay, R. (2024). Multiverse: A Deep Learning 4X4 Sudoku Solver. In *ICAART (3)* (pp. 15-22).

[29] Pal, A., Chandra, S., Mongia, V., Behera, B. K., & Panigrahi, P. K. (2020). Solving Sudoku game using a hybrid classical-quantum algorithm. *Europhysics Letters*, *128*(4), 40007.

[30] Hasanah, N. A., Atikah, L., Herumurti, D., & Yunanto, A. A. (2020, September). A comparative study: Ant colony optimization

algorithm and backtracking algorithm for sudoku game. In *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)* (pp. 548-553).IEEE. https://doi.org/10.1109/iSemantic50169.2020.9234267

[31] Jana, S., Dey, A., Maji, A. K., & Pal, R. K. (2021). A novel hybrid genetic algorithm-based firefly mating algorithm for solving sudoku. *Innovations in Systems and Software Engineering*, *17*(3), 261-275. https://doi.org/10.1007/s11334-021-00397-4

[32] Jana, S., Mallik, M., Khan, A., Maji, A. K., & Pal, R. K. (2023). Design and analysis of a modified 3D sudoku solver. *IEEE Access*, *11*, 27352-27368. https://doi.org/10.1109/ACCESS.2023.3256420

[33] Schottlender, M. (2014, May). The effect of guess choices on the efficiency of a backtracking algorithm in a Sudoku solver. In *IEEE Long Island Systems, Applications and Technology (LISAT) Conference 2014* (pp. 1-6). IEEE.

[34] Khazaei, S., & Moazami, F. (2017). On the Computational Complexity of Finding a Minimal Basis for the Guess and Determine Attack. *ISeCure*, *9*(2).

[35] Abin, A. A., & Vu, V. V. (2020). A density-based approach for querying informative constraints for clustering. *Expert Systems with Applications*, *161*, 113690. https://doi.org/10.1016/j.eswa.2020.113690

[36] Altbawi, S. M. A., Khalid, S. B. A., Mokhtar, A. S. B., Shareef, H., Husain, N., Yahya, A., ... & Alsisi, R. H. (2023). An improved gradient-based optimization algorithm for solving complex optimization problems. *Processes*, *11*(2),498. https://doi.org/10.3390/pr11020498

[37] Ananya, G. M., & Singh, A. K. (2022). Augmented Reality Sudoku Solver. *ECS Transactions*, *107*(1), 2085. https://doi.org/10.1149/10701.2085ecst

[38] Dugar, A., Sohanraj, R., Agarwal, S., Salvi, S., & Borkar, M. (2024, July). Augmented Reality-based Sudoku Solver with Training Module to Improve Cognitive Skills. In *2024 IEEE 3rd World Conference on Applied Intelligence and Computing (AIC)* (pp. 60-66). IEEE.

[39] Tsai, P. S., Wu, T. F., Chen, J. Y., & Huang, J. F. (2022). Integrating of image processing and number recognition in Sudoku puzzle cards digitation. *Journal of Internet Technology*, *23*(7), 1573-1584. https://doi.org/10.53106/160792642022122307012

[40] Sitorus, P., & Zamzami, E. M. (2020, June). An implementation of backtracking algorithm for solving a Sudoku-Puzzle based on Android. In *Journal of Physics: Conference Series* (Vol. 1566, No. 1, p. 012038). IOP Publishing. https://doi.org/10.1088/1742-6596/1566/1/012038

[41] Syed, A. T., Merugu, S., & Kumar, V. (2020). Augmented reality on sudoku puzzle using computer vision and deep learning. In *Advances in Cybernetics, Cognition, and Machine Learning for Communication Technologies* (pp. 567-578). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-15-3125-5_55

[42] Pranav, G., Varsha, K., Nazar, S. A., Prasannan, V. P., & Sharmila, T. S. (2025, March). Backtracking algorithm with optical character recognition for solving sudoku puzzles. In *AIP Conference Proceedings* (Vol. 3137, No. 1, p. 020041). AIP Publishing LLC.

[43] Ciantar, K. G., & Casha, O. (2023, July). Implementation of a Sudoku Puzzle Solver on a FPGA. In *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)* (pp. 1803-1808). IEEE. https://doi.org/10.1109/CoDIT58514.2023.10284457

[44] Gao, S., Xiang, S., Song, Z., Han, Y., & Hao, Y. (2021). All-optical Sudoku solver with photonic spiking neural network. *Optics Communications*, *495*, 127068. https://doi.org/10.1016/j.optcom.2021.127068

[45] Pignari, R., Fra, V., Macii, E., & Urgese, G. (2025). Efficient solution validation of constraint satisfaction problems on neuromorphic hardware: the case of Sudoku puzzles. *IEEE Transactions on Artificial Intelligence*. https://doi.org/10.1109/TAI.2025.3536428

[46] Anasune, A., & Bhavsar, S. (2023, November). Image based Sudoku Solver using Applied Recursive Backtracking. In *2023 2nd International Conference on Futuristic Technologies (INCOFT)* (pp. 1-5). IEEE.

[47] Najafian, M., Gulliver, T. A., & Esmaeili, M. (2022). Construction of standard solid Sudoku cubes and 3D Sudoku Puzzles. *IEEE Access*, *10*, 30180-30188. https://doi.org/10.1109/ACCESS.2022.3159027

[48] Brown, J. (2024). Sudoku strategies using graph theory. *The Mathematical Gazette*, *108*(572), 275-282. https://doi.org/10.1017/mag.2024.68

[49] Maria Jeyaseeli, J., Lau, G. C., Shiu, W. C., & Arumugam, S. (2023). Sudoku number of graphs. *AKCE International Journal of Graphs and Combinatorics*, *20*(2), 209-216. https://doi.org/10.1080/09728600.2023.2218917

[50] Chu, L., & Chen, H. (2025). On the tightness of graph-based statistics. *Electronic Journal of Statistics*, *19*(1), 1425-1452. https://doi.org/10.1214/25-EJS2367

[51] Prates, M., & Lamb, L. (2018, November). Problem solving at the edge of chaos: Entropy, puzzles and the sudoku freezing transition. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 686-693). IEEE.

[52] Ates, T., & Cavdur, F. (2025). Sudoku puzzle generation using mathematical programming and heuristics: Puzzle construction and game development. *Expert Systems with Applications*, 127710. https://dl.acm.org/doi/abs/10.1016/j.eswa.2025.127710

[53] D'Agostino, M. (2013). Depth-bounded logic for realistic agents. *L&PS–Logic & Philosophy of Science*, *11*(1), 3-57.

[54] Diah, N. M., Riza, S., Ahmad, S., Musa, N., & Hashim, S. (2025). Sudoku Solutions: A Comparative Analysis of Breadth-First Search, Depth-First Search, and Human Approaches. *Journal of Education and Learning (EduLearn)*, *19*(1), 561-569.

[55] Kalia, V., Fuesting, M., & Cody, M. (2019). Perseverance in solving Sudoku: Role of grit and cognitive flexibility in problem solving. *Journal of Cognitive Psychology*, *31*(3), 370-378.

[56] Behrens, T., Räuker, M., Kalbfleisch, M., & Jäkel, F. (2023). Flexible use of tactics in Sudoku. *Thinking & Reasoning*, *29*(4), 488-530. https://doi.org/10.1080/13546783.2022.2091040

[57] Meng, T., & Chang, K. W. (2021, July). An integer linear programming framework for mining constraints from data. In *International Conference on Machine Learning* (pp. 7619-7631). PMLR.

[58] Serradilla, O., Zugasti, E., Rodriguez, J., & Zurutuza, U. (2022). Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects. *Applied Intelligence*, *52*(10), 10934-10964. https://doi.org/10.1007/s10479-021-04441-3

[59] Nabwey, H. A., Ashraf, M., Nadeem, H., Rashad, A. M., & Chamkha, A. J. (2024). Optimizing renewable energy systems: A comprehensive review of entropy generation minimization. *AIP Advances*, *14*(12). https://doi.org/10.1063/5.0245560

[60] Robert, L., Miyahara, D., Lafourcade, P., Libralesso, L., & Mizuki, T. (2022). Physical zero-knowledge proof and NP-completeness proof of Suguru puzzle. *Information and Computation*, *285*, 104858. https://doi.org/10.1016/j.ic.2021.104858

[61] Sungur, B., & Madenoğlu, F. S. (2025). Integer Programming Formulations For Generating And Solving Survo Puzzle. *Erciyes Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi*, (71), 41-44.

[62] Indriyono, B., Pamungkas, N., Pratama, Z., Mintorini, E., Dimentieva, I., & Mellati, P. (2023). Comparative analysis of the performance testing results of the backtracking and genetics algorithm in solving Sudoku games. *International Journal of Artificial Intelligence & Robotics (IJAIR)*, *5*(1), 29-35.

[63] Lloyd, H., & Amos, M. (2019). Solving Sudoku with ant colony optimization. *IEEE Transactions on Games*, *12*(3), 302-311. https://doi.org/10.1109/TG.2019.2942773

[64] Chatzinikolaou, T. P., Karamani, R. E., Fyrigos, I. A., & Sirakoulis, G. C. (2024). Handling Sudoku puzzles with irregular learning cellular automata. *Natural Computing*, *23*(1), 41-60. https://link.springer.com/article/10.1007/s11047-024-09975-4

[65] Thangamani, R., & Regulagedda, P. (2026). A Computational Complexity-Based Framework for Predicting the Difficulty of Sudoku Puzzles Using CSP-Based Symbolic Modeling and Constraint Metrics. Journal of Computational and Cognitive Engineering. https://doi.org/10.47852/bonviewJCCE52026137

[66] Rajan, T., & Pallavi, R. (2024). The Study on Predictive Maintenance in Industry 4.0 with Complexity Analysis Ideas of AI-Algorithms. Nanotechnology Perceptions, 20(S1), 61–73. https://nano-ntp.com/index.php/nano/article/view/389