Simen Owen Academic Proceedings Series

4th International Conference on Electronics, Engineering, Computer Science and Applied Development (EESD 2026)

Article

Sudoku Explain: Auditable Search Traces and a Reproducible Python Solver Comparison

Zihan Shi[1,*]

[1] International Department, Zhongshan Sanxin School, Zhongshan, China
* Correspondence: Zihan Shi, International Department, Zhongshan Sanxin School, Zhongshan, China

Abstract: Sudoku provides a compact setting for studying constraint satisfaction and heuristic search, but an answer grid alone does not reveal which decisions were logically forced and which depended on assumptions. This paper presents Sudoku Explain, a local Python application that records rule applications, speculative assignments, contradictions, and branch restoration for interactive replay. The implementation combines established minimum-remaining-values selection with naked and hidden singles; it does not introduce a new solving algorithm. A separate set-based exact-cover implementation checks solution uniqueness. A controlled comparison evaluates four configurations on 60 generated puzzles with 44, 36, or 28 clues, using three repetitions per puzzle and configuration. All 720 runs returned valid solutions within the specified budgets. In the 28-clue group, median runtimes were 84.115 ms for row-order search and 1.808 ms for the full configuration; the median paired speedup was 39.39. A separate experiment audited 180 recorded traces, covering 8,196 assignment events, and measured a median recording-time ratio of 1.045 relative to unrecorded runs. These findings establish measured behavior for this implementation and synthetic dataset, rather than general solver superiority or educational effectiveness. The artifact provides an inspectable starting point for learning about symbolic AI and for extending the evaluation to harder external benchmarks.

Keywords: Sudoku; constraint satisfaction; heuristic search; explanation traces; reproducibility; educational software

Received: 08 September 2026
Revised: 17 October 2026
Accepted: 01 November 2026
Published: 07 November 2026

Copyright: © 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

A standard Sudoku puzzle asks for a completion of a partially filled 9 by 9 grid such that each row, column, and 3 by 3 box contains the digits 1 through 9 exactly once. It is a useful small problem for connecting data structures, recursion, constraint checking, and heuristic decisions. Constraint-based Sudoku models and their solution methods are established [1, 2]. The educational interest of implementing a solver therefore lies in understanding and exposing the reasoning, rather than claiming that automatically filling the grid is itself a research novelty.

An execution display can nevertheless be misleading. A digit assigned after a guess may be forced only within that branch. A later contradiction can invalidate both the guess and all deductions below it. Displaying only assignments that survive in the final grid hides this distinction and may turn a search trace into an apparent sequence of unconditional deductions. The design question addressed here is how a small implementation can retain these distinctions and make its behavior testable.

This work asks two bounded research questions. RQ1: on a fixed synthetic dataset, how do runtime and search behavior change as MRV selection, naked-single propagation, and hidden-single propagation are added to a common solver? RQ2: can the resulting

execution traces be replayed and checked against explicit rule witnesses, and what recording overhead is observed? The system is intended to support learning, but no claim about improved learning outcomes is evaluated.

The deliverables are a Python implementation, a local Chinese-language interface, a replay audit, an independent uniqueness checker, and a reproducible experimental package. The contribution is an implementation and evaluation artifact. It is neither the first visual Sudoku solver nor the first work on explaining Sudoku solutions.

2. Related Work

Simonis models Sudoku as a constraint problem [1]. Norvig's technical essay gives a compact implementation combining propagation and search [2]. These sources provide background for the present design; Norvig's essay is an implementation reference rather than a peer-reviewed experimental comparison. The minimum-remaining-values heuristic belongs to the established family of fail-first variable selection methods studied in constraint search [3].

The most direct visual precedent is the interactive Sudoku system of Howell et al [4]., which illustrates consistency algorithms and their use in education and outreach. Björnsson et al [5]. go beyond exposing execution: they search for solutions using familiar human techniques and a model of explanatory effort. The present system does not optimize cognitive cost, implement their full rule sets, or replicate their evaluations. It records the actual deterministic execution of a much smaller collection of rules, including unsuccessful branches.

Exact cover provides a structurally different formulation for verification. Algorithm X and the efficient Dancing Links representation are described by Knuth [6]. The verifier here implements Algorithm X using Python sets and copied mappings. It does not implement Dancing Links, and its runtime is not included in the solver comparison. These earlier works establish both the algorithmic foundations and the limited novelty of the current artifact.

3. Model and Solver Configurations

3.1. Constraint Model

For each cell i, let x_i be an integer in {1,...,9}. The 27 units consist of nine rows, nine columns, and nine boxes. Each unit requires all its variables to be different, and every clue fixes one variable. For a partial board b, the local candidate set of an empty cell i is

C(i,b) = {1,...,9} minus {b_j : j shares a row, column, or box with i and b_j is nonzero}.

The implementation recomputes these sets from the current board. This favors readability over incremental domain maintenance. A nonempty local candidate set does not prove that a complete extension exists. Conversely, an empty set establishes that the current branch cannot yield a solution.

3.2. Rules and Search

A naked single occurs when the candidate set of an empty cell has size one. A hidden single occurs when a missing digit has exactly one candidate location within a unit. The full configuration applies naked singles first. When none is available, it scans units in row, column, then box order, and digits in ascending order, for a hidden single. After an assignment, candidates are recomputed. If a missing digit has no location within a unit, the full configuration reports a contradiction.

When propagation stops short of a solution, a variable is selected and its candidates are tried in ascending order. MRV chooses the cell with the fewest candidates; ties use row-major index. Each child branch receives a board copy. A failed child is discarded, and the trace records restoration to the parent board. Search terminates at the first solution or when the available branches are exhausted. Operational time and node budgets can interrupt this process; an interrupted search reports an unresolved status, not unsatisfiability.

The four configurations share candidate computation, duplicate-clue validation, and empty-domain checking:

Code	Variable selection	Explicit propagation
Row	First empty cell in row-major order	None
MRV	Minimum candidate count	None
MRV+N	MRV after propagation	Naked singles
MRV+N+H	MRV after propagation	Naked and hidden singles; unit no-location check

The Row baseline is thus row-order backtracking with an all-empty-cell candidate check, not an unpruned enumeration. MRV may select a cell with one candidate, even without an explicit propagation loop. Its forced assignment is counted as a search assignment. Adding hidden-single propagation also adds the unit no-location contradiction check, so that comparison does not isolate only successful hidden-single assignments. The design is a sequential component comparison rather than a complete factorial experiment over selection and propagation.

3.3. Independent Uniqueness Checking

Finding one completion is insufficient to establish uniqueness. The separate verifier maps each possible cell-digit assignment to four exact-cover constraints: one value per cell, and one occurrence of each digit per row, column, and box. This produces 324 columns. Candidate rows incompatible with clues are omitted. The set-based Algorithm X search chooses a remaining column of minimum size and recursively covers compatible rows.

Counting stops at two solutions. Two witnesses establish multiple solutions; exhausting the search with one establishes uniqueness; exhausting it with zero establishes unsatisfiability. If the time budget expires before either conclusion is available, the status is unknown. The verifier does not import the main solver's candidate or peer structures, reducing shared implementation risk. This is cross-checking, not a formal correctness proof.

3.4. Trace Semantics and Interface

Each event stores its type, the board after the event, the affected cell and value when applicable, the candidates before an assignment, the active assumption depth, a unit identifier where relevant, and a short explanation. Event types distinguish naked singles, hidden singles, speculative assignments, forced search assignments, contradictions, backtracks, and completion. Assumption depth counts active decisions with more than one candidate, rather than all recursive calls.

The interface labels a speculative assignment as a trial. Logical assignments at positive depth explicitly depend on the current assumptions. A backtrack snapshot restores the parent board, including removal of all assignments in the failed child. The viewer can step backward and forward through the recorded states, play them automatically, jump along a timeline, or export JSON. It is a replay of a completed bounded search rather than a live stream of ongoing computation. Candidate witnesses shown for an assignment refer to the pre-assignment state.

The replay auditor checks that each assignment changes only its target cell, respects its recorded candidates, and satisfies the stated single rule when applicable. It maintains a stack of search trials and verifies backtrack restoration. Completion events must satisfy the Sudoku constraints and preserve clues. The auditor reuses the solver's candidate calculation, so it is not fully independent; the exact-cover check separately verifies the number of completions of the original puzzle. The auditor does not validate the assumption-depth field or the natural-language explanation strings, and it does not certify a proof of unsatisfiability. The software tests also exercise non-duplicate unsatisfiable input, malformed boards, multiple solutions, completed boards, deterministic behavior, and search limits.

The application uses Python's standard-library HTTP server bound to the loopback address and a small HTML/CSS/JavaScript interface. Its runtime requires no external package or network service. Full snapshots simplify replay at the cost of storing 81 cell values per event. The application caps recorded events at 15,000 and visibly reports truncation; the solver result remains separate from the truncated event list. A truncated trace is not accepted as a complete audit target.

4. Experimental Method

4.1. Dataset

Sixty puzzles were generated locally, with 20 target puzzles each at 44, 36, and 28 clues. For each puzzle, randomized backtracking first constructs a complete board. Randomness selects among tied MRV cells and shuffles candidate values. A shuffled order of cells is then considered for clue removal. A removal is retained only if the independent exact-cover checker confirms uniqueness within its budget. Generation stops when the target count is reached or the removal order is exhausted. All 60 generated puzzles reached their targets in this run.

The base seed is 20261027. Seeds are base + 1000g + j, for group g in {0,1,2} and index j in {0,...,19}. The exact puzzle strings, full reference grids, seeds, and clue counts are supplied in the dataset. Literal duplicate puzzle strings were checked and absent. The generator does not sample uniformly from Sudoku puzzles and does not canonicalize symmetry classes. Clue count is a grouping variable, not a validated label of human difficulty. No puzzle was excluded based on the measured solver performance. The dataset SHA-256 is df52d00d35ecdb5e97002ad79120b6dfd61249d2527fecf213e16a9865d9d633.

4.2. Timing and Metrics

Experiments ran on an Apple M2 machine with macOS 15.6.1 and CPython 3.9.6. Every puzzle was solved by every configuration three times, producing 720 runs. The job order was shuffled within each repetition using seed 20261028. Each configuration received one untimed call on a completed grid before measurement. This is a minimal warmup, not a comprehensive stabilization procedure. The host was an ordinary desktop environment, not a dedicated benchmark machine.

The solver's internal perf_counter interval excludes input parsing, uniqueness verification, JSON serialization, browser rendering, and file writing. Event collection was disabled for the main comparison. Each run used a one-second limit and a 100,000-node limit, checked cooperatively during search. Failed or budget-limited runs remain in the raw data. Every returned completion was checked for unit validity and preservation of clues; generated reference solutions were also checked in the test suite.

A node is an entry into the recursive search function. A guess is an attempted value at a cell with more than one candidate. A backtrack is a failed value trial, not the number of erased cells. Deductions count explicit single-rule applications, including those later undone. Because propagation can perform many assignments within one node, node counts are not directly comparable units of computational work across configurations.

For each puzzle and configuration, the three elapsed times are reduced to a median. Group summaries report the median and interquartile range across these 20 puzzle-level medians. Paired speedup is calculated for each puzzle as Row time divided by comparison time, then summarized across the same puzzles. Repeated timings are not treated as additional independent puzzles. The analysis is descriptive; no significance claim or population confidence interval is reported.

4.3. Trace Experiment

A separate experiment compares MRV+N+H with event collection enabled and disabled on the same 60 puzzles, with three repetitions and order seed 20261029. This gives 360 runs. Every enabled trace is audited after the timed solve; JSON size is measured after timing as well. The experiment therefore measures recording work inside the solver,

excluding auditing, serialization, transmission, and display. It is not an end-to-end interface latency measurement. For each puzzle, three times are reduced to a median separately for each condition; the ratio of these two medians is then summarized across puzzles.

5. Results

5.1. Solver Behavior

All four configurations solved all 60 puzzles in all repetitions: 720 valid solutions, no budget-limited runs, and no invalid completions. Consequently, the runtime summaries use the same puzzles for every configuration. Table 1 presents runtime medians and interquartile ranges. Figure 1 shows the distributions of per-puzzle medians on a logarithmic time axis.

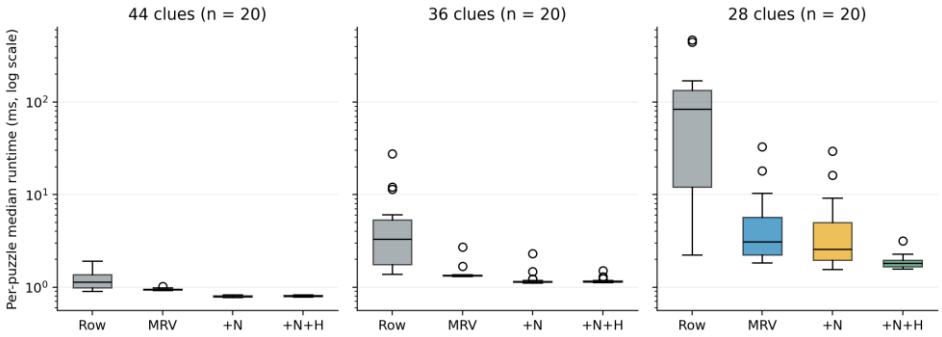


Figure 1. Runtime Distributions. Row: Row-Order Search; +N: MRV with Naked Singles; +N+H: MRV with Both Single Rules. Boxes Represent the Middle 50% of Puzzle-Level Medians; Whiskers Extend to Points within 1.5 IQR.

Table 1. Runtime in Milliseconds; Median [Q1, Q3] Across 20 Puzzle-Level Medians Per Group. Each Puzzle-Level Median Contains Three Runs.

Clues	Row	MRV	MRV+N	MRV+N+H
44	1.137 [0.989, 1.362]	0.942 [0.936, 0.957]	0.792 [0.782, 0.814]	0.806 [0.797, 0.822]
36	3.291 [1.759, 5.342]	1.338 [1.322, 1.358]	1.140 [1.132, 1.166]	1.154 [1.137, 1.175]
28	84.115 [12.127, 134.618]	3.087 [2.227, 5.712]	2.564 [1.951, 5.012]	1.808 [1.658, 1.964]

The largest separation occurs in the 28-clue group. Its median paired speedup relative to Row is 14.58 for MRV, 17.73 for MRV+N, and 39.39 for MRV+N+H. The last figure differs from 84.115/1.808 because the median of paired ratios is not generally the ratio of medians. These are observed implementation-level comparisons, not speedups over optimized general-purpose solvers (As shown in Table 2).

Table 2. Median Guesses / Failed Value Trials in the 28-Clue Group; Counts Include Unsuccessful Search Branches.

Row	MRV	MRV+N	MRV+N+H
1414.5 / 2275	8 / 35	8 / 3	0 / 0

Backtracks can outnumber guesses because trials at cells with one candidate can also lead to failure downstream. MRV and MRV+N have equal median ambiguous trial counts in this group, while their recursive structure and failed-trial counts differ. This illustrates why no single counter should stand in for runtime.

In the 44- and 36-clue groups, MRV+N+H has slightly higher median runtime than MRV+N. The study does not establish a meaningful difference at this small scale: these are sub-millisecond or millisecond measurements on a shared desktop. It would be unwarranted to claim that every extra propagation rule always improves speed. Across all groups, the full configuration used any guess on only three of the 60 puzzles and

backtracked on one. The dataset therefore provides limited stress on complex branch explanation.

5.2. Trace Audit and Overhead

All 180 recorded MRV+N+H runs passed the replay audit, covering 8,196 assignment events. These are three repeated executions per puzzle, not 180 distinct puzzles or 8,196 independent validations of a theorem. Traces contained a median of 47 events, with a range of 39 to 91, and a median serialized size of 19,119 UTF-8 bytes. No trace reached the recording cap.

The median paired ratio of recorded to unrecorded solver time was 1.045, with an interquartile range of [1.037, 1.052]. This corresponds to about 4.5% additional median recording time under the stated timing boundary. Snapshot memory, browser drawing, JSON encoding, and longer traces may have different costs. Thirteen automated tests also passed, including interface request validation and trace edge cases; this test evidence does not establish complete behavioral coverage.

6. Limitations and Next Steps

The strongest limitation is the dataset. The 60 synthetic puzzles share one generation procedure, are not drawn uniformly, and lack independently assigned difficulty ratings. Most are solved by the full configuration without branching. Results should not be generalized to adversarial puzzles, all clue counts, or human performance. An extension should add external datasets with clear provenance and separate training or design examples from evaluation examples if any learned component is introduced.

The solver is deliberately simple. It recomputes candidate sets and copies boards, while the verifier copies exact-cover mappings. It omits advanced human rules and optimized SAT, CSP, or Dancing Links baselines. The sequential comparison also leaves interactions unresolved: adding a propagation-plus-row-order configuration would help distinguish selection and propagation effects. Unit no-location detection should be separately ablated if the contribution of hidden singles alone is the research target.

Timing uses one machine, three repetitions, and an ordinary desktop workload. The low runtimes make relative differences sensitive to interpreter and system overhead. No formal significance or hardware-independent performance claim follows. The trace audit shares candidate code with the solver, and a passing test suite is weaker than a formally verified implementation. The current auditor uses Python assertions and must be run without the -O optimization flag. Additional mutation tests, independent witness checkers, and external benchmark cross-checks would strengthen reliability evidence.

Finally, readable explanations are a design aim, not a measured user outcome. No participants, questionnaires, learning scores, or classroom intervention are included. A future study could compare a completed-grid display with conditional trace replay using predefined comprehension tasks, appropriate consent, and a suitable study design. Until then, educational benefit remains a hypothesis.

7. Conclusion

Sudoku Explain combines established constraint search techniques with explicit records of deductions, assumptions, contradictions, and rollback. The supplied implementation makes the path to a solution inspectable and provides an independent uniqueness check. On the fixed synthetic benchmark, stronger configurations reduced observed search and runtime, while trace recording imposed modest measured overhead. The result is a reproducible student research artifact with stated limits, rather than a new algorithm, a universal performance result, or evidence of improved learning.

References

1. H. Simonis, "Sudoku as a Constraint Problem," in *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, 2005. [Online]. Available: https://research.ucc.ie/en/publications/sudoku-as-a-constraint-problem/

2. P. Norvig, "Solving Every Sudoku Puzzle," 2006. [Online]. Available: https://www.norvig.com/sudoku.html. Accessed: Oct. 27, 2026.
3. R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, 1980. doi: 10.1016/0004-3702(80)90051-X.
4. I. Howell, R. Woodward, B. Y. Choueiry, and C. Bessiere, "Solving Sudoku with Consistency: A Visual and Interactive Approach," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 5829–5831, 2018. doi: 10.24963/ijcai.2018/852.
5. Y. Björnsson, S. Helgason, and A. Pálsson, "Searching for Explainable Solutions in Sudoku," in *2021 IEEE Conference on Games (CoG)*, pp. 1–8, 2021. doi: 10.1109/CoG52621.2021.9618900.
6. D. E. Knuth, "Dancing Links," in *Millennial Perspectives in Computer Science*, pp. 187–214, 2000. [Online]. Available: https://doi.org/10.48550/arXiv.cs/0011047.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of Publisher and/or the editor(s). Publisher and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.