Scaling Sudoku as a Constraint Problem

Mikael Z. Lagerkvist
Independent Researcher, `zayenz.se`, Stockholm, Sweden
SambaNova Systems, Stockholm, Sweden

Abstract

Helmut Simonis's 2005 ModRef paper, *Sudoku as a Constraint Problem*, used standard 9x9 Sudoku as a compact constraint-programming laboratory for comparing propagation-oriented models on published instances. This paper extends that setup to Sudoku puzzles of sizes 6x6, 9x9, 16x16, 25x25, and 36x36. We generate 32,000 unique-solution base puzzles and run 170,000 hardness walks that add clues back to the puzzles, generating 402,201 additional, simpler instances.

The original hardness categories identified by Simonis remain useful, but the generated corpus changes regime with grid size. The 6x6, 9x9, and 16x16 starting instances are dominated by different propagation buckets. In contrast, none of the 1,000 generated 25x25 starting puzzles or the 1,000 generated 36x36 starting puzzles is solved without search by the tagged propagation family, even with domain-consistent propagation and domain shaving. The walks show that these larger instances are not isolated from easier buckets, but the average number of added clues per verified bucket drop rises from 1.20 at 6x6 to 24.12 at 36x36.

Puzzle generation uses Gecode and OR-Tools, and hardness testing uses Gecode.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Sudoku, constraint modelling, propagation, hardness analysis, benchmark instances

Acknowledgements The author thanks the anonymous reviewers for their helpful suggestions.

1 Introduction

Simonis used Sudoku because it is small but not trivial: rows, columns, and boxes give a compact all-different model, while small modelling changes can still alter how much propagation proves before search is needed [11]. This paper keeps that framing and changes one thing: the size of the grid.

Changing the size exposes behaviour that 9x9 alone can hide. Generalized Sudoku is well known to be computationally hard [13], but there has not been a larger systematic test of how Sudoku behaves as it scales. In the generated instance set, 6x6, 9x9, and 16x16 form three distinct propagation regimes, while 25x25 and 36x36 are not solved by any of the current propagation solution levels before clue additions. A second experiment, hardness walks, adds back clues from the solution to produce easier instances. These walks support the same directed acyclic graph of hardness categories as Simonis's paper.

The difference from Simonis's experiment is the source and scale of the instances. Simonis used standard 9x9 Sudoku as a modelling laboratory and used published puzzle instances to investigate and illustrate the propagation hierarchy. This paper generates unique-solution instances across five sizes, targets local minimality, and then tags the generated instances with the Simonis-style propagation schemes.

Contributions. The contributions are threefold. First, we describe a Sudoku generation implementation for unique-solution instances across several grid sizes, with local minimality as the default construction target. Second, we use it to study how the observed Simonis hardness levels change as Sudoku grows from 6x6 and 9x9 to 16x16, 25x25, and 36x36. This includes checking the expected implication lattice between propagation solution levels:

Table 1 Hardness-label vocabulary used in the paper. The labels name the weakest propagation configuration that solves the puzzle without search; `Search` means that none of the tagged propagation configurations did so.

Label part	Meaning
`Val`, `Bnd`, `Dom`	Base propagation level for the Sudoku all-different model: value propagation, bounds-consistent propagation, or domain-consistent propagation.
`BS`	Bounds shaving added to the base propagation level.
`DS`	Full-domain singleton shaving added to the base propagation level.
`Chan`	Value-occurrence channeling constraints for rows, columns, and boxes.
`Same`	Row/block and column/block count-equality constraints.
`Cmat`	A colored-matrix decomposition using value indicators and row/column occurrence constraints.
`Rbm`	Row-band/block and column-stack/block matching constraints.
`Search`	No tagged propagation scheme solved the instance without search. The puzzle is still a unique-solution puzzle.

whenever a puzzle is solved by a given scheme, the expected stronger schemes also solve it. Third, we provide a generated benchmark corpus: 32,000 unique-solution base instances and 170,000 clue-adding hardness walks, producing 402,201 saved simpler walk-step variants. The corpus is intended for experimental evaluation of scaling behaviour and may also support learning-benchmark work, motivated by recent use of Sudoku variants in reasoning benchmarks [12, 3, 10].

Roadmap. Sections 2 and 3 describe the modelling vocabulary and generated data. Section 4 gives the static hardness result. Section 5 covers the hardness-walk analysis. Sections 6 and 7 discuss benchmark use and conclusions.

2 Background and Setup

We study finite-domain Sudoku variants defined by a grid size and a box shape. Standard 9x9 Sudoku uses 3x3 boxes. The generated instance set also includes 6x6 puzzles with 3x2 boxes,[1] 16x16 with 4x4 boxes, 25x25 with 5x5 boxes, and 36x36 with 6x6 boxes. Every row, column, and box is constrained to contain distinct values; in a CP implementation, this makes the all-different propagator central [8].

The hardness axis follows Simonis's Sudoku hierarchy [11]. Each puzzle is checked against a fixed family of propagation schemes, and the analysis keeps the full set of schemes that solve the instance without search. The primary label is the weakest successful scheme. If no tagged scheme succeeds without search, the paper reports the fallback bucket as `Search`. The label is coarse, but it is explicit, reproducible, and comparable across sizes. Table 1 summarizes the label vocabulary used in the rest of the paper. It is normal for pen-and-paper puzzles to require a deductive path to a solution [1], and therefore the `Search` bucket represents either instances that are not suitable as puzzles or instances for which this tagging family is missing some reasoning.

[1] The 6x6 case is also recognizable outside benchmark collections: LinkedIn launched a 3x2-box Mini Sudoku as a daily game in 2025, reportedly with millions of daily players [4, 6].

The labels are compositional. A name such as `DomBS` means that the domain-propagation model solves the puzzle after adding bounds shaving. A name such as `DomSameCmatRbm` means that the domain model is combined with the `Same`, `Cmat`, and `Rbm` redundant model constraints. The label reported for a puzzle is the weakest successful member of this family; the analysis also keeps the full set of successful schemes so that implication checks can be made after tagging.

The `Cmat` variant used here follows Simonis's colored-matrix model vocabulary. It is related to the cardinality matrix constraint of Régin and Gomes [9], but the implementation used in these experiments is a decomposition, not a dedicated global cardinality-matrix propagator.

3 Generating Instances

The generator is built as a C++ application that produces puzzles, validates uniqueness and local minimality, computes hardness tags, and emits benchmark instances for the sizes studied here. It uses Gecode [2] for solution generation and Simonis-style hardness tagging, and OR-Tools CP-SAT [7] for uniqueness checks. MiniZinc-based hooks are available for cross-checking selected runs [5]. The result is a generated benchmark set whose instances can be compared by size, clue density, propagation hardness, and local response to added clues. The implementation separates the construction problem from the tagging problem: generation tries to remove as many clues as possible while keeping a unique solution, and tagging later asks which propagation configuration, if any, solves the resulting puzzle without search.

Instance generation starts from a complete solved grid. The generator builds a full solution with the Sudoku all-different model in Gecode, then removes clues in a randomized order while preserving uniqueness by searching for up to two solutions using a CP-SAT model. The two-solution check blocks the first solution before looking for another one, so it does not establish uniqueness by repeatedly finding the known complete grid.[2] Removal is tried in larger batches first and then refined toward single-clue checks. A removal is kept only when the remaining puzzle still has a unique solution; otherwise the clue is restored. By default the final pass targets local minimality, meaning that no single remaining clue can be removed without losing uniqueness. The uniqueness check has a timeout. As a result, the 36x36 instances are unique-solution puzzles but usually do not have an exact local-minimality certificate.

The Simonis hardness tags reported below are computed by the Gecode propagation configurations, including the shaving variants. CP-SAT is used for part of generation and validation, not the implementation of the `BS` or `DS` hardness levels.

A hardness walk starts from a tagged puzzle and repeatedly adds clues from the known solution. At each step, the walk chooses among still-empty cells weighted by the unresolved cells of propagation schemes just below the current hardness bucket (fewer remaining choices are preferred). After each added clue, the relevant schemes are updated incrementally. When the walk detects that an easier bucket is solving the puzzle, it verifies the new bucket by rerunning the propagation schemes in order and records the resulting variant. The walk stops when it reaches `Val`.

[2] A separate uniqueness-check variant searched directly for a solution that is *different* from the known complete grid. It did not improve runtime, so the implementation uses the two-solution check.

Table 2 Generated benchmark inventory. The Walk variants column reports saved simpler instances produced by the hardness walks. The `Exact local minimality` column reports the share of starting instances with an exact local-minimality certificate.

Size	Box	Puzzles	Walks	Walk variants	Exact local minimality
6x6	3x2	10,000	50,000	49,515	100.0%
9x9	3x3	10,000	50,000	92,456	100.0%
16x16	4x4	10,000	50,000	167,026	100.0%
25x25	5x5	1,000	10,000	46,079	100.0%
36x36	6x6	1,000	10,000	47,125	0.2%

The generated benchmark contains 32,000 unique-solution starting instances: 10,000 each for 6x6, 9x9, and 16x16, plus 1,000 each for 25x25 and 36x36. It also contains 170,000 hardness walks: 50,000 each for 6x6, 9x9, and 16x16, plus 10,000 each for 25x25 and 36x36. These walks produced 402,201 simpler variants in addition to the original 32,000 instances.

4 Hardness Distribution Across Sizes

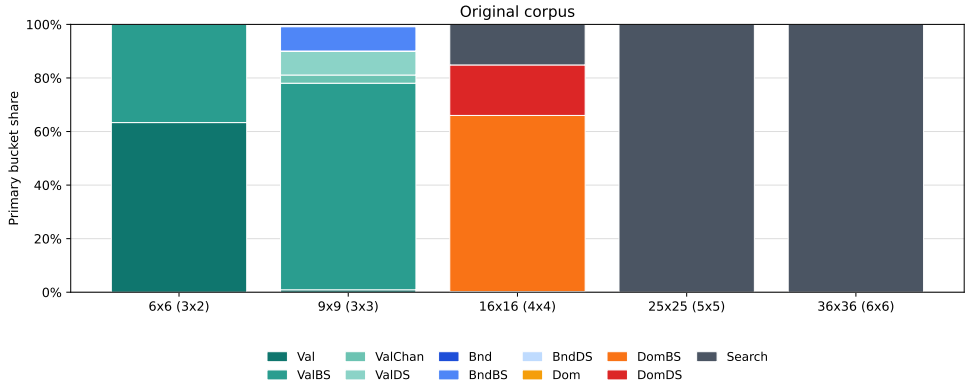
In the base generated instance set, 6x6 is dominated by `Val` (63.3%), 9x9 by `ValBS` (77.1%), and 16x16 by `DomBS` (65.8%). Propagation-only coverage is complete for 6x6 and 9x9, drops to 84.9% at 16x16, and is not observed at 25x25 and 36x36. Under this propagation family, 0 of the 1,000 generated 25x25 starting instances and 0 of the 1,000 generated 36x36 starting instances are solved without search.

Table 3 Cross-size static hardness summary. Prop.-only rate is the share of instances solved by at least one tagged propagation scheme without search. Median fill is the share of filled cells in the starting instance.

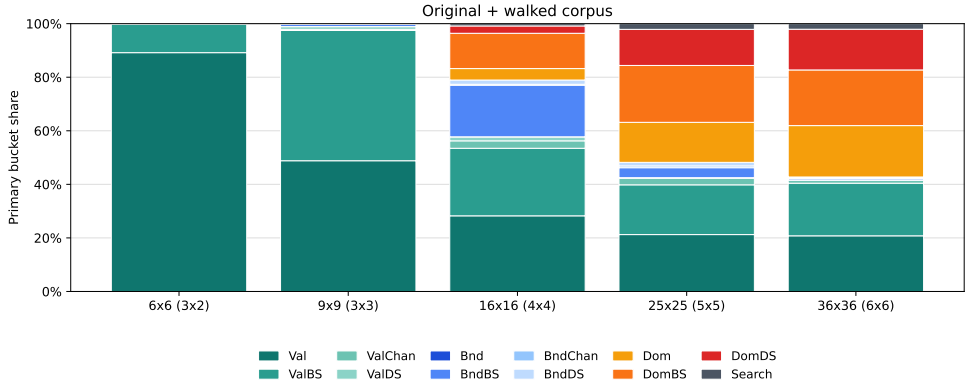
Size	Dominant bucket	Dominant share	Prop.-only rate	Median clues	Median fill
6x6	Val	63.3%	100.0%	10	27.8%
9x9	ValBS	77.1%	100.0%	24	29.6%
16x16	DomBS	65.8%	84.9%	94	36.7%
25x25	Search	100.0%	0.0%	269	43.0%
36x36	Search	100.0%	0.0%	677	52.2%

Standard 9x9 is, as expected, harder than 6x6, but it is still fully covered by propagation alone. At 16x16 the mass moves into domain-propagation buckets and a visible `Search` tail appears. At 25x25 and 36x36 the current propagation family no longer solves any starting instance without search. The combined distribution in Figure 1 adds all walk variants: the generated data gain substantial `Val` and `ValBS` mass, but the larger sizes remain visibly different from 6x6 and 9x9. The combined plot is walk-biased rather than a natural corpus distribution: a source puzzle can contribute several saved variants, and sources with longer walks therefore contribute more observations.

There are three limits on this interpretation. First, the experiment studies the distribution produced by this generator, not all generalized Sudoku instances. Second, clue density is not controlled independently of size: the median fill rises from 27.8% at 6x6 to 52.2% at 36x36 because these are the densities produced by the uniqueness-preserving removal process. Third, the 36x36 instances are unique-solution instances, but only 0.2% have an



(a) Generated instance set.



(b) Generated plus walk variants.

Figure 1 Primary hardness-bucket distribution by size in the generated instance set and after adding all hardness-walk variants. The generated-instance plot gives the static result. The combined plot shows the easier buckets reached by local clue additions.

exact local-minimality certificate under the timeout-limited final pass. The 36x36 evidence therefore describes this large generated unique-solution regime, not exact-local-minimality scaling in general.

For each puzzle, the analysis records all propagation schemes that solve it without search, not only the weakest one. The expected hierarchy holds for the generated instances we can test: whenever a puzzle is solved by a given propagation scheme, it is also solved by the schemes expected to be stronger in the Simonis lattice. We do not make a claim about parts of the hierarchy where no generated puzzle is solved by the weaker scheme.

5 Hardness Walks

Every walk in the experiment reaches `Val`. The sizes differ in how much evidence the walk must add before verified bucket drops occur. The average number of added clues per verified drop rises from 1.20 in 6x6 to 3.13 in 9x9, 8.42 in 16x16, 15.30 in 25x25, and 24.12 in 36x36. Runtime follows the same broad pattern, rising from 17.6 ms per 6x6 walk to 314 s per 36x36 walk.

Table 4 Hardness-walk summary. Added clue mass is the sample-weighted mean share of source empty cells added per verified bucket drop.

Size	Walks	Reached `Val`	Hints/drop	Clue mass/drop	Mean time
6x6	50,000	100.0%	1.20	4.8%	17.6 ms
9x9	50,000	100.0%	3.13	6.5%	162.1 ms
16x16	50,000	100.0%	8.42	9.0%	5.07 s
25x25	10,000	100.0%	15.30	9.8%	75.11 s
36x36	10,000	100.0%	24.12	9.0%	314.09 s

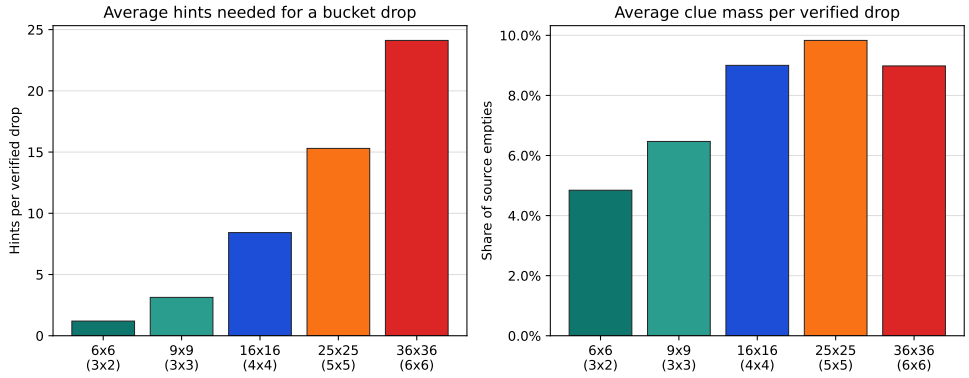


Figure 2 Cross-size walk cost summary. The left panel shows the mean number of added clues needed for a verified hardness drop; the right panel normalizes that quantity by the source puzzle's empty cells. Larger instances require more absolute clues and a larger local perturbation before the bucket changes.

The walks are guided descents. They do not sample all possible clue additions; they test whether a nearby easier instance can be reached by adding true-solution clues chosen from the current propagation state.

In Simonis's paper [11] there are puzzles that fall into the domain propagation plus extra constraints buckets, `DomChan`, `DomSame`, `DomCmat`, and `DomRbm`. No generated instance matches these buckets, and no walk lands in them either. The generator optimizes uniqueness and local minimality, not membership in a particular propagation bucket. It therefore has no reason to preserve the structural patterns that make a puzzle sit exactly in `DomChan`, `DomSame`, `DomCmat`, or `DomRbm`. The walks then add true-solution clues and move locally toward easier instances. They are useful for finding nearby easier variants, but they are not a search over the full hardness lattice and can bypass narrow regions of it. This suggests a qualitative difference between targeted or human-designed puzzle instances and the generated instances studied here.

The full set of hardness directed acyclic graphs is given in Appendix A.

6 Benchmark Outlook

The generated benchmark set is available at `zayenz/scaled-sudoku-instances`. The repository contains the starting puzzles and the saved walk variants. Each instance is paired with a sidecar file containing the solution, generation metadata, the primary hardness label, and the full set of propagation schemes that solved the instance without search. This makes

the corpus usable either as a collection of Sudoku instances or as tagged input data for experiments that need size, clue density, and propagation hardness as explicit fields. There is script support for generating symmetric variants of puzzles, as well as individual instances along the generated walks. As Table 4 shows, the average number of clues added for 36x36 is more than 24, which means that one can create more than 24 closely related instances per level on average.

Many existing Sudoku collections are designed for human solving, solver stress tests, or fixed 9x9 evaluation. The corpus here is meant for a narrower comparison: it gives unique-solution instances across several grid sizes, explicit Simonis-style propagation tags, and hardness-walk neighbours obtained by adding clues from the solution. It is not meant to replace existing collections, or to make a diversity claim against them.

Sudoku has reappeared in reasoning benchmarks for neural systems, including recursive models and curated Sudoku-variant tasks [12, 3, 10]. These papers use 9x9 Sudoku instances to train or test neural reasoning. The results here suggest a natural follow-on question: whether those methods scale with the underlying reasoning structure when the grid size increases, rather than only with the number of input cells.

7 Conclusion

This paper gives an empirical account of Sudoku beyond 9x9 using one generated instance set and the Simonis-style hardness analysis. The main result is that size changes the propagation regime. Across 32,000 generated instances, 6x6 is dominated by `Val`, 9x9 by `ValBS`, 16x16 shifts into domain-style buckets with a visible `Search` tail, and 25x25 and 36x36 are entirely in `Search` under the current propagation family.

The hardness walks give the local-neighbourhood part of the result. The large instances are not isolated from the propagation-only region: every walk in the experiment reaches `Val` by adding clues from the solution. What changes with size is the local perturbation needed for verified bucket drops. The average number of added clues per drop rises from 1.20 at 6x6 to 24.12 at 36x36, and the corresponding walk runtimes rise from milliseconds to minutes.

The result is also a benchmark artifact: 32,000 unique-solution starting instances, plus walk variants that cover several hardness levels across the studied sizes. The expected Simonis hierarchy holds for the generated instances we can test: whenever a puzzle is solved by one propagation scheme, the schemes expected to be stronger also solve the puzzle. Starting at 16x16, however, search-free solving is no longer complete, and for 25x25 and 36x36 `Search` is the only observed starting bucket before clue additions simplify the instances.

For future work, one priority is to look for better Sudoku models that solve some of the `Search` instances. Investigating the unreached hardness buckets and finding a way to generate instances for them would also help clarify how human-designed puzzle instances differ from generated ones. Another direction is to test solvers with strong pre-processing, such as OR-Tools CP-SAT, to see how their pre-processing relates to this propagation lattice, or whether it reveals other hardness categories.

References

1 Joan Espasa, Ian P. Gent, Ruth Hoffmann, Christopher Jefferson, Matthew J. McIlree, and Alice M. Lynch. Towards generic explanations for pen and paper puzzles with MUSes? In *Proceedings of the SICSA eXplainable Artificial Intelligence Workshop 2021*, volume 2894 of *CEUR Workshop Proceedings*, pages 56–63. CEUR-WS.org, 2021. URL: `https://ceur-ws.org/Vol-2894/short8.pdf`.

2 Gecode Team. Gecode: Generic constraint development environment. https://www.gecode.dev/, 2026. Accessed 2026-05-01.

3 Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. arXiv:2510.04871, doi:10.48550/arXiv.2510.04871.

4 LinkedIn Corporate Communications. Linkedin announces mini sudoku: Our 6th thinking-oriented game in collaboration with the magazine that handcrafted it. https://news.linkedin.com/2025/linkedin-announces-minisudoku-, 2025. Accessed 2026-05-12.

5 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7_38.

6 Jordan Novet. Linkedin launches mini sudoku, pushing deeper into casual games that keep users coming back. https://www.cnbc.com/2025/08/12/linkedin-mini-sudoku-games.html, 2025. Accessed 2026-05-12.

7 Laurent Perron, Frédéric Didier, and Steven Gay. The cp-sat-lp solver. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:2. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CP.2023.3.

8 Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 362–367. AAAI Press, 1994. URL: https://cdn.aaai.org/AAAI/1994/AAAI94-055.pdf.

9 Jean-Charles Régin and Carla P. Gomes. The cardinality matrix constraint. In *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 572–587. Springer, 2004. doi:10.1007/978-3-540-30201-8_42.

10 Jeffrey Seely, Yuki Imajuku, Tianyu Zhao, Edoardo Cetin, and Llion Jones. Sudoku-bench: Evaluating creative reasoning with sudoku variants, 2025. arXiv:2505.16135, doi:10.48550/arXiv.2505.16135.

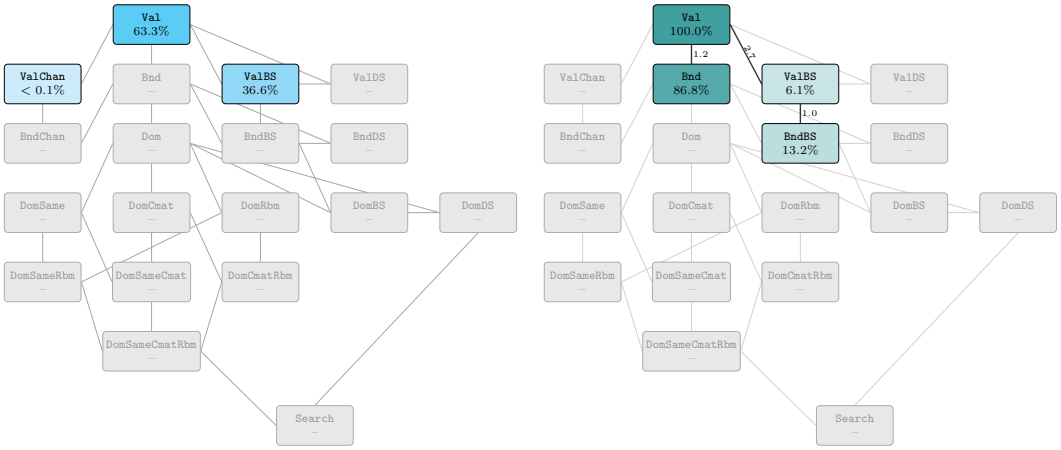
11 Helmut Simonis. Sudoku as a constraint problem. In *Modelling and Reformulating Constraint Satisfaction Problems*, volume 3769 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2005. doi:10.1007/11564751_2.

12 Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi-Yadkori. Hierarchical reasoning model, 2025. arXiv:2506.21734, doi:10.48550/arXiv.2506.21734.

13 Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A(5):1052–1060, 2003.

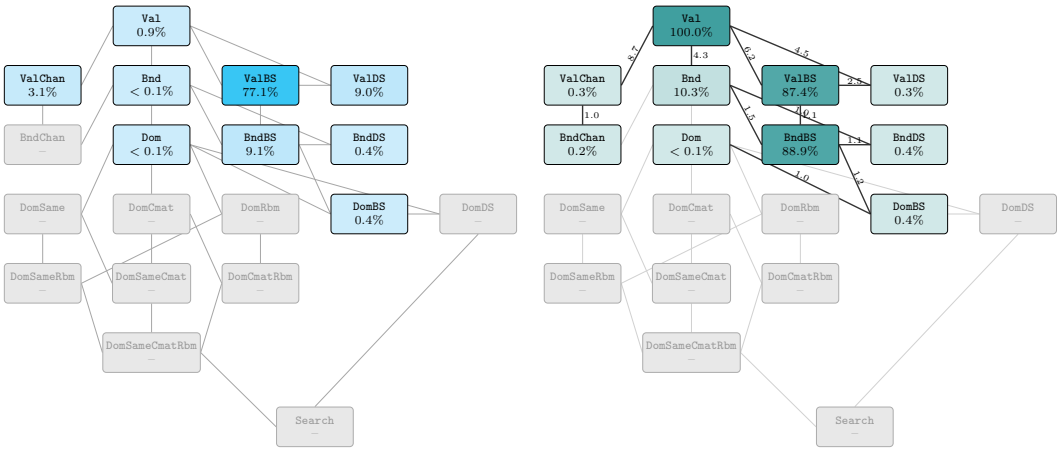
A Hardness DAG Diagrams

This appendix shows the full hardness DAG diagrams for the generated instance set and the walk-variant set. In each pair, the left diagram shows the static propagation-lattice occupancy for the generated instances, and the right diagram shows the hardness-walk transition lattice. Node labels report the share of the relevant data that occupies or reaches the node; walk edge labels report the mean number of added clues on observed transitions.



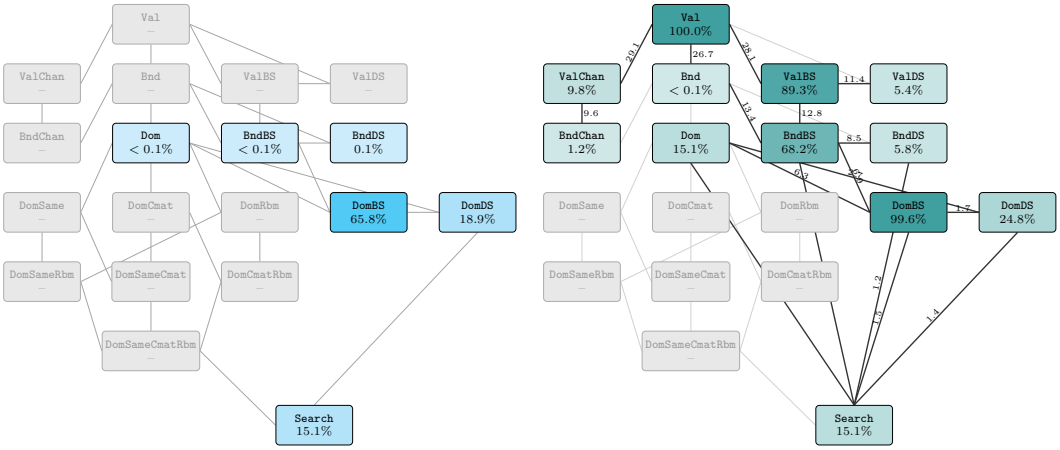
(a) Static occupancy. **(b)** Walk transitions.

Figure 3 Hardness DAG diagrams for the 6x6 generated data.



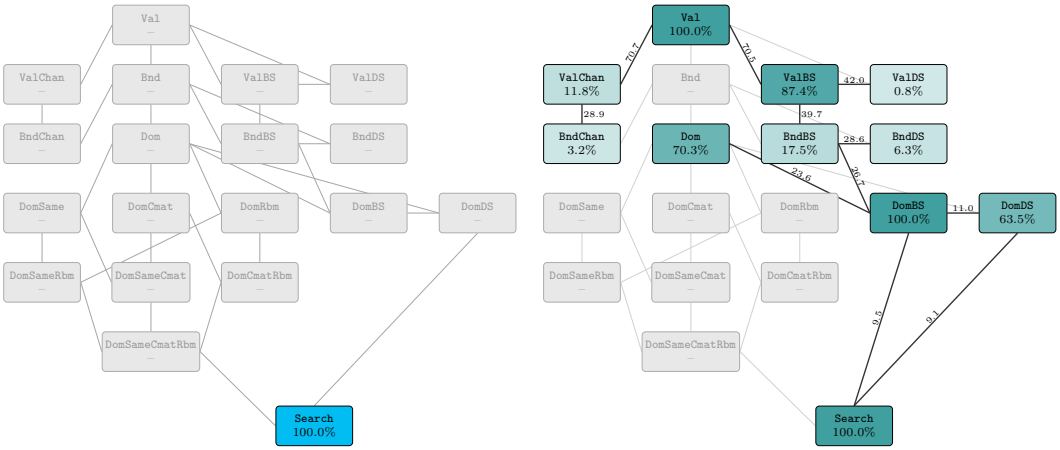
(a) Static occupancy. **(b)** Walk transitions.

Figure 4 Hardness DAG diagrams for the 9x9 generated data.



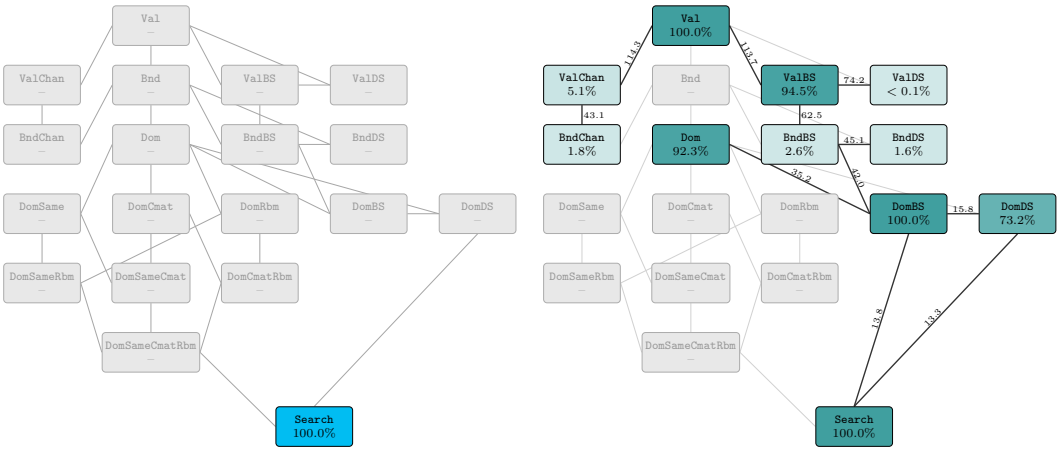
(a) Static occupancy. **(b)** Walk transitions.

Figure 5 Hardness DAG diagrams for the 16x16 generated data.



(a) Static occupancy. **(b)** Walk transitions.

Figure 6 Hardness DAG diagrams for the 25x25 generated data.



(a) Static occupancy. **(b)** Walk transitions.

Figure 7 Hardness DAG diagrams for the 36x36 generated data.