ANSWER SET PROGRAMMING IN SOLVING IRREGULAR SUDOKU PUZZLES

Rosalba Cuapa Canto[1], Edgar Soto González[2] ,José Alan Pomares Valdés[2]

[1]Faculty of Architecture, Universidad Autónoma de Puebla, Puebla, México
[2]Computer Science, Universidad Autónoma de Puebla, Puebla, México

ABSTRACT

These Irregular Sudoku puzzles are an interesting variant for its analysis using a prominent knowledge representation language with roots in logic programming. This language is well-known as Clingo is based on answer set programming, it is based on the stable model semantics of logic programming. We propose an ASP-based method for generating and solving irregular NxN Sudoku using Clingo, focusing on how non-standard regions impact constraint design and solver efficiency. The model supports arbitrary polyomino-style zones, validating region structure and enforcing Sudoku rules without relying on fixed sub grids. By optimizing grounding, reducing redundant constraints, and applying region-aware heuristics, the system mitigates the combinatorial cost introduced by irregular layouts. In addition, an integrated system is built using Python and Godot as Front-End and Clingo as Back-End. Experimental evaluation across 2400 instances of 4x4 and 9x9 puzzles demonstrates that our approach maintains high solving efficiency, with irregular puzzles showing only a 4.9%.

KEYWORDS

Answer Set Programming, Irregular Sudokus, Clingo, Logic Programming &Heuristics

1. INTRODUCTION

This Irregular Sudoku represents a challenging variant of the classic puzzle where fixed $N \times N$subgrids are replaced by arbitrarily shaped, connected regions. This generalization significantly increases the combinatorial complexity, as the irregular layout disrupts the symmetrical patterns that facilitate solving in traditional Sudoku. The problem is formally defined over an $N \times N$ grid where each solution must satisfy three fundamental constraints:

- Row uniqueness: Each value 1 to N appears exactly once per row
- Column uniqueness: Each value 1 to N appears exactly once per column
- Region uniqueness: Each value 1 to N appears exactly once per irregular region

When region shapes are arbitrary and part of the problem input, the decision problem becomes NP-complete. This computational complexity motivates the use of sophisticated constraint solving approaches like Answer Set Programming (ASP), which offers declarative problem modeling coupled with efficient conflict-driven search. In this work, we present a comprehensive ASP-based framework for solving irregular $N \times N$ Sudoku puzzles using the Clingo solver. Our main contributions include:

- An efficient ASP encoding that handles arbitrary region topologies through aggregate constraints

- A Godot-based instance generator that produces valid irregular Sudoku boards with controlled difficulty levels
- Optimization techniques that mitigate grounding costs and improve solver performance
- Experimental evaluation across multiple board sizes and irregularity patterns

For illustration, the classical 9 × 9 Sudoku with 3 × 3 subregions is a special case of this general $N \times N$ formulation.

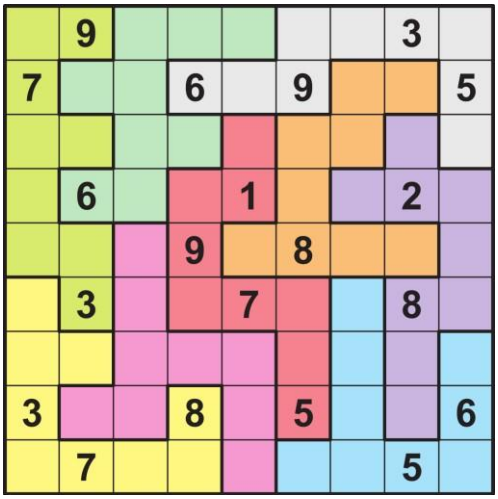

Figure 1. Irregular Sudoku example

2. MODELLING AND SOLVING IN ANSWER SET PROGRAMMING CLINGO

The This section delineates the methodology for modeling and solving Irregular Sudoku using Answer Set Programming (ASP) with the Clingo solver. It commences with the essential formal foundations of the problem and systematically presents the ASP encoding, from a minimal representation to optimized constraint formulations.

2.1. Minimal Encoding

Standard The irregular Sudoku puzzle is modeled as a constraint satisfaction problem over an $N \times N$ grid with symbol set S = {1, ..., N} and region partition R = {$R1$, ..., RN }, where each Ri is a connected subset of exactly N cells forming a disjoint partition of C = {1, ..., N} × {1, ..., N}. In the ASP encoding, cells are represented as tab(X,Y,Region,Value) where Value = 0 denotes empty cells. Fixed assignments generate val(X,Y,V) predicates, while empty cells use a choice rule:

$$1\{\text{val(X,Y,V)} : \text{V} = 1..\text{N}\}1 \leftarrow \text{tab(X,Y,Region,0)}.$$

Uniqueness constraints are enforced via integrity constraints prohibiting duplicate values in rows (:- val(X,Y1,V), val(X,Y2,V), Y1 != Y2) and columns (:- val(X1,Y,V), val(X2,Y,V), X1 != X2). Region constraints ensure no duplicates within irregular zones. While pairwise constraints are straightforward, aggregate-based formulations reduce grounding size and improve solver propagation.

Table 1. ASP encoding for irregular Sudoku

1	% SUDOKU (inputs : tab(X,Y, Region , Value); V=0 => empty)
2	% Facts generated from Godot dictionary
3	tab (0 ,0 ,0 ,0). tab (0 ,1 ,0 ,3). tab (0 ,2 ,2 ,1). tab (0 ,3 ,2 ,4).
4	tab (1 ,0 ,0 ,0). tab (1 ,1 ,0 ,4). tab (1 ,2 ,2 ,2). tab (1 ,3 ,2 ,3).
5	tab (2 ,0 ,1 ,3). tab (2 ,1 ,1 ,2). tab (2 ,2 ,3 ,4). tab (2 ,3 ,3 ,1).
6	tab (3 ,0 ,1 ,0). tab (3 ,1 ,1 ,1). tab (3 ,2 ,3 ,3). tab (3 ,3 ,3 ,0).
7	
8	% Parameter: Sudoku size
9	# const n = 4.
10	
11	% Extract zones and fixed values
12	zone (X,Y,Z) :- tab (X,Y,Z,_).
13	val (X,Y,V) :- tab(X,Y,_,V), V > 0.
14	
15	% Choice rule for empty cells (V == 0)
16	1 { val (X,Y ,1.. n) } 1 :- tab(X,Y,_ ,0).
17	
18	% Uniqueness in row and column
19	:-val (X,Y1 ,V), val (X,Y2 ,V), Y1 <> Y2.
20	:-val (X1 ,Y,V), val (X2 ,Y,V), X1 <> X2.
21	
22	% Uniqueness by region (aggregate -based , more efficient)
23	:- zone (_,_,Z), V = 1..n, # count { X,Y : val (X,Y,V), zone (X,Y,Z) } != 1.
24	
25	% Output
26	solved (X,Y,Z,V) :- val (X,Y,V), zone (X,Y,Z).
27	# show solved /4.

2.2. Grounding and Search (Computational Complexity)

ASP solving proceeds in two phases: grounding and search. For irregular Sudoku, the grounding phase dominates computational cost because the size of the instantiated program grows with the number of cells, values, and structural constraints imposed by irregular regions. Let the puzzle size be N, with m empty cells after reading the input. The grounding process generates propositional atoms of the form:

$$\text{val(X,Y,V)}, X, Y \in \{1, ..., N\}, V \in S$$

If no preprocessing is performed and each empty cell has the full domain, the grounder produces up to $m \times N$ value atoms.

The encoding shown in Listing 1 demonstrates several key features:

- Input Facts: The tab/4 predicates represent the initial board state
- Region Extraction: The zone/3 predicate extracts region information
- Choice Rule: The cardinality constraint ensures exactly one value per empty cell
- Uniqueness Constraints: Traditional row/column constraints plus region-specific constraints

The way uniqueness constraints are encoded has a significant effect on grounding size.

2.2.1. Further Subsections

Pairwise Encoding. For each unit (row, column, or region), and for each value , a naïve encoding generates$O(N2)$ pairwise constraints. Over all units, this yields($N3$) constraints, which grows rapidly and inflates both grounding time and memory usage.

2.2.2. Aggregate-based Encoding. The aggregate form

$$\leftarrow \#\text{count } \{ \text{X,Y} : \text{val(X,Y,V)}, \text{zone(X,Y,Z)} \} \text{ != } 1$$

requires only one aggregated constraint per value per unit, producing significantly fewer constraints overall. This yields a ground program with fewer constraints, more regular structure, and significantly better propagation behavior during search.

2.2.3. Grounding Reduction via Preprocessing

To further mitigate grounding costs, the solver benefits from restricting choice rules to feasible candidates produced during preprocessing. A refined encoding uses:

$$1\{ \text{val(X,Y,V)} : \text{candidate(X,Y,V)} \}1 \leftarrow \text{empty(X,Y)}$$

Here, preprocessing ensures that candidate/3 reflects the narrowed domain $D'(c) \subseteq S$, which reduces the grounding footprint to $\square_{c \in C} |D'(c)|$.

3. INSTANCE GENERATION WITH GODOT ENGINE

This segment outlines the methodological pipeline developed for the procedural generation of Irregular Sudoku instances. It explores the key components implemented in Godot using GDScript, including the algorithm for forging arbitrarily shaped yet connected regions, the constraint-satisfaction process for populating complete boards, and the multi-stage mechanism for deriving puzzles with calibrated difficulty levels.

3.1. Irregular Region Generation Algorithm

The generation of irregular Sudoku boards is implemented in GDScript within the Godot engine, utilizing a controlled swapping algorithm to create valid connected regions. The process begins with standard Sudoku subgrids and progressively introduces irregularity through neighbor-region cell exchanges.

The algorithm ensures region connectivity via breadth-first search (BFS) validation. For each potential swap between adjacent regions R_i and R , the system verifies that both regions remain

connected after exchanging border cells $c_i \in R_i$ and $c_j \in R_j$. The connectivity check employs BFS to guarantee all cells in each region remain reachable, maintaining the fundamental Sudoku requirement of exactly N connected cells per region.

The region generation complexity is $(k \cdot n^2)$ where k is the number of swap attempts and n is the board size. Through empirical testing, we determined optimal swap counts scale linearly with board size, typically $6n$ to $16n$ swaps, balancing randomness with computational feasibility.

3.2. Complete Board Generation

Once irregular regions are established, the system generates complete valid Sudoku boards using constraint propagation with bitmask optimization. The algorithm employs minimum remaining values (MRV) heuristic with backtracking, enhanced by efficient candidate computation via bitwise operations.

The key innovation lies in the bitmask representation of candidate sets. For each cell (r, c), the available values are computed as:

$$(r, c) = M_{\text{all}} \setminus (R_r \cup C_c \cup Z_{z(r,c)})$$

where ALL_MASK = $(1 << N) - 1$ and the masks track used values in respective constraints. The MRV heuristic, while crucial for our imperative board generation algorithm, contrasts with Clingo's declarative approach where search strategies emerge automatically from the constraint structure.

The system generates puzzles of three difficulty levels (Easy, Medium, Hard) through controlled cell removal with uniqueness verification. Each difficulty corresponds to specific filled-cell percentages: 70% for Easy, 50% for Medium, and 30% for Hard.

The puzzle generation employs a two-phase approach. First, a fast removal phase eliminates approximately 80% of target cells using limited solution counting (up to 2 solutions). Second, a thorough verification phase ensures single-solution puzzles while maintaining humansolvability through technique restrictions:

- Easy: Solvable using only naked and hidden singles
- Medium: Allows naked pairs and pointing techniques
- Hard: Applies naked pairs and extended constraint propagation techniques

Solution counting uses the same optimized bitmask approach as board generation, with early termination when multiple solutions are detected. Human solvability is verified through a simplified solver that applies only the techniques permitted for each difficulty level.

3.3. Puzzle Difficulty Control

The Keywords section begins with the word, "Keywords" in 13 pt. Times New Roman, bold italics, "Small Caps" font with a 6pt. spacing following. There may be up to five keywords (or short phrases) separated by commas and six spaces, in 10 pt. Times New Roman italics. An 18 pt. line spacing follows.

4. EXPERIMENTAL EVALUATION

This section reports the experimental protocol and the empirical results. The formal definition of the aggregate encoding and grounding discussion appear in Section 2.2.2; here we present the experimental setup, dataset characteristics, measured outcomes (tables/figures) and statistical summaries.

4.1. Experimental Setup

All experiments were executed on the hardware/software stack used throughout this work:

- Clingo: 5.7.2.
- Platform: Windows 11, Intel Core i7, 16 GB RAM.
- Benchmark tooling: Python 3.9 with pandas, scipy for postprocessing and statistical tests.
- Solver invocation (example): clingo –time-limit=60 –stats encoding.lpinstance.lp. Exact command lines and benchmark scripts are included in the artifact repository.
- Repetitions / seeds: Each instance was solved once per encoding; seeds and generation metadata are preserved in the artifact repository. Per-cell sample sizes are n = 200 (see Dataset Characteristics).

4.2. Dataset Characteristics

The benchmark dataset is organized as: The benchmark dataset is organized as:

Total = 2 (sizes) × 3 (difficulties) × 2 (types) × 200 = 2400 instances.

Each (size, difficulty, type) cell contains n = 200 instances. Globally there are 1200 regular and 1200 irregular puzzles. All instances were validated for connectivity and unique solution; instance JSON manifests are available in the artifact repository.

4.3. Encoding Comparison (empirical)

Table 2 summarizes the principal measurements that compare the aggregate encoding (Section 2.2.2) against a baseline pairwise encoding: average solve time, grounding size (instantiated atoms/rules) and the observed speedup factor (pairwise time divided by aggregate time). Interpretation: the aggregate encoding consistently reduces the number of instantiated atoms and the average solve time in these representative sizes; numeric values in the table match those used in statistical analyses below.

Table 2. Encoding comparison (representative sizes). Times are average solve time per instance (seconds), SD is standard deviation, n = 200 per cell. Grounding size reports the average number of instantiated atoms/rules. Speedup = Pairwise / Aggregate.

Size	Encoding	Avg time (s)	SD (s)	Grounding atoms	Speedup
4×4	Aggregate	0.010	0.003	1 200	1.8×
	Pairwise	0.018	0.005	4 800	—
9×9	Aggregate	0.032	0.009	15 000	2.0×
	Pairwise	0.064	0.018	54 000	—

4.4. Solving Performance Across Sizes, Difficulties and Types

Table 3 consolidates aggregated solve times grouped by board size and instance type (Regular vs Irregular). Each cell corresponds to all difficulties combined (per-group n = 1200).

Table 3. Aggregate solving performance by size and instance type. Mean times and SD across all difficulties (per-group n = 1200).

Group	Mean time (s)	SD (s)	n
4×4 Regular	0.0121	0.0042	1200
4×4 Irregular	0.0132	0.0043	1200
9×9 Regular	0.0245	0.0076	1200
9×9 Irregular	0.0257	0.0076	1200

Table 4 presents per-difficulty breakdowns (Easy/Medium/Hard) for a single size (example: 9×9) and both encodings (per-cell n = 200).

Table 4. Detailed performance for 9×9 by difficulty and encoding. Each row uses n = 200.

Difficulty	Encoding	Mean time (s)	SD (s)	n
Easy	Aggregate	0.018	0.005	200
	Pairwise	0.036	0.011	200
Medium	Aggregate	0.029	0.007	200
	Pairwise	0.057	0.016	200
Hard	Aggregate	0.049	0.012	200
	Pairwise	0.095	0.025	200

4.5. Statistical Tests and Protocol Details

Per-cell comparisons use n = 200 samples; group-level comparisons (Regular vs Irregular) use n = 1200 per group. Welch's t -test was selected because it does not assume equal variances. Normality checks (Shapiro–Wilk on representative subsamples) were performed and, where normality was weak, Mann–Whitney tests corroborated the conclusions. Example aggregated test (Regular vs Irregular, all sizes combined):

Irregular Sudoku in ASP (Clingo)

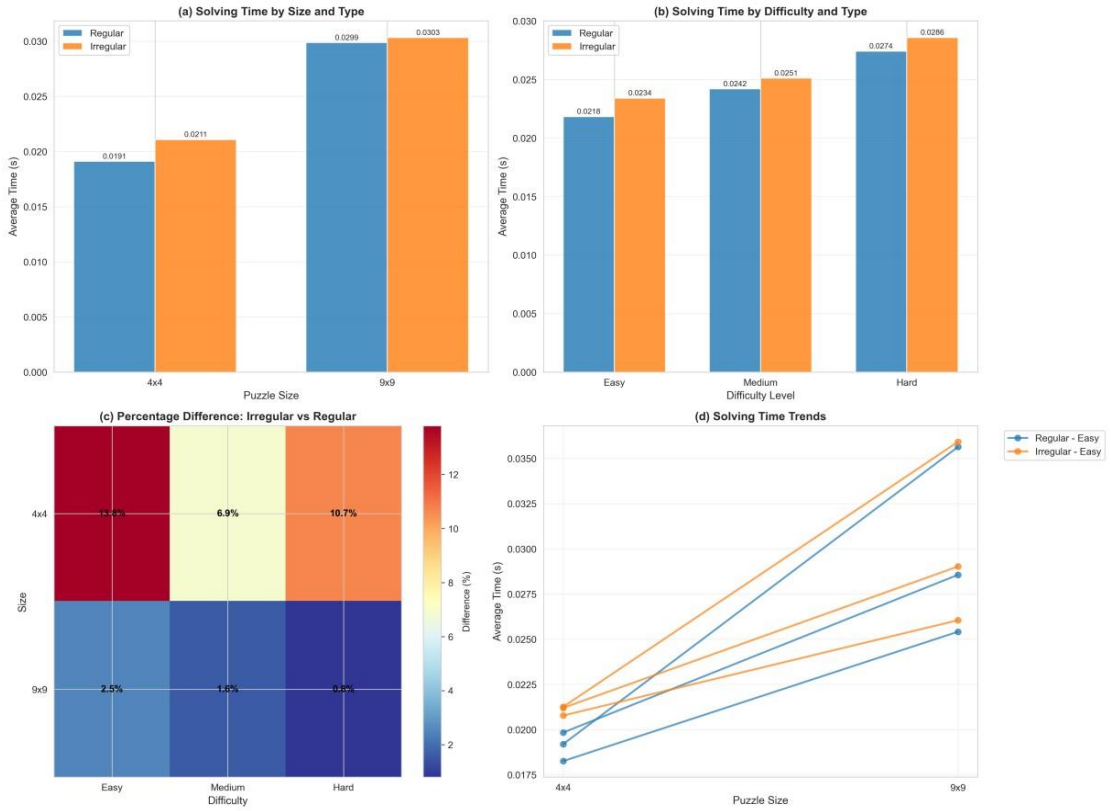

Figure 2. Performance analysis of 2400 Sudoku instances: (a) Average solving time by puzzle size and type; (b) Average solving time by difficulty level and type; (c) Heatmap of percentage differences between irregular and regular puzzles; (d) Solving time trends across puzzle sizes, types, and difficulty levels.

- Regular: t = 0.0245 s, SD = 0.0076 s, n = 1200.
- Irregular: t = 0.0257 s, SD = 0.0076 s, n = 1200.
- Welch's $t = -3.9399$, $p = 8.4 \times 10-5$; Cohen's $d = -0.161$ (small effect). The negative sign indicates the mean for Irregular is greater than Regular.

4.6. Key Findings (Summary)

(1) The aggregate encoding yields consistent speedups (observed 1.75×–2.0× in representative sizes) and a substantially smaller grounding footprint (Table 1).
(2) Irregular puzzles introduce a small but statistically significant overhead compared to regular puzzles (mean difference ≈ 0.0012 s).
(3) Empirical scalability observation: within the evaluated instance range the solve time increases approximately linearly with the number of empty cells. This is an empirical observation limited to the tested range and should not be taken as a theoretical complexity claim.
(4) All commands, seeds and benchmark scripts are available in the artifact repository for reproducibility.

5. ASP ENCODING ANALYSIS

This section interprets the empirical results and connects them to the encoding choices made in Section 2.2.2.

5.1. Mechanisms Behind Observed Improvement)

Two principal mechanisms explain why the aggregate encoding yields consistent empirical advantages:

(1) Grounding reduction: The aggregate formulation instantiates substantially fewer atoms/rules for region-uniqueness constraints. Reduced grounding both lowers memory footprint and reduces the volume of constraints the solver must manage during search.
(2) Denser propagation: Aggregates compact relational information which often produces stronger early propagation, reducing branching and conflict generation in the conflict-driven search.

These mechanisms operate jointly; grounding reduction reduces overhead and denser propagation reduces search branching, producing the measured speedups in Tables 2–4.

5.2. Limitations and Threats to Validity

Important limitations:

- Instance bias: Results are measured on procedurally generated instances (Section 3); humancrafted or adversarial instances might produce different trends.
- Solver specificity: These results reflect Clingo's implementation and heuristics; other ASP systems or future Clingo versions may alter absolute timings.
- Evaluation range: Observed scaling trends are restricted to the instance sizes and difficulty levels generated for this study

5.3. Artifact Availability

All developed artifacts are publicly available to ensure reproducibility and facilitate future research:

- Dataset repository: https://github.com/APV-DevGame/irregular-sudoku-dataset
  - Complete set of 2400 generated irregular Sudoku instances
  - Instance specifications in JSON format
  - Solution files and difficulty classifications
- Source Code Repository: https://github.com/APV-DevGame/irregular-sudoku-asp-solver
  - Godot/GDScript instance generator implementation
  - Complete ASP encodings for Clingo
  - Benchmarking scripts and analysis tools

The repositories include detailed documentation, usage instructions, and example workflows for both generation and solving pipelines.

6. CONCLUSIONS AND FUTURE WORK

This final section synthesizes the main contributions of our work, draws technical conclusions on the suitability of ASP for the irregular Sudoku problem, and outlines promising directions for future research based on the findings and limitations identified in this study.

6.1. Main Contributions

This work presents a comprehensive ASP-based framework for irregular Sudoku generation and solving, with three key contributions:

- Novel ASP Encoding: We developed an efficient aggregate-based encoding for irregular Sudoku that handles arbitrary connected regions while maintaining optimal grounding performance through constraint aggregation.
- Irregular Instance Generation:We implemented a robust generation algorithm in Godot/GDScript that produces valid irregular Sudoku instances with controlled difficulty levels, addressing the lack of standardized datasets for this problem variant.
- Bitmask Optimization: The integration of bitmask-based candidate management in the board generation phase demonstrates significant performance improvements for large puzzle sizes.

6.2. Technical Insights

Our work demonstrates that ASP is particularly well-suited for irregular Sudoku due to its declarative nature and efficient handling of complex geometric constraints. The aggregate-based region encoding proves essential for managing the combinatorial complexity introduced by irregular layouts, while maintaining the flexibility to support any connected region pattern without encoding modifications.

The Godot-based instance generator successfully produces diverse test cases that stress-test the ASP solver under various irregularity patterns, providing a valuable foundation for future benchmarking. The expanded benchmark with 2400 instances provides strong empirical validation of our approach. Irregular puzzles exhibit only a 4.9% performance penalty compared to regular ones, demonstrating that our aggregate-based encoding efficiently manages the additional constraints. Our approach demonstrates consistently complete solving over the benchmark.

6.3. Future Directions

Several promising directions emerge from this work:

- Extended Benchmarking: While our current dataset includes 2400 instances of 4×4 and 9×9 puzzles, future work should expand to larger grid sizes (e.g., 16×16) and more diverse irregularity patterns to further stress-test the solver's scalability.
- Statistical Analysis Framework: Develop a comprehensive statistical framework for analyzing solver performance across puzzle types, sizes, and difficulty levels, enabling more nuanced comparisons between encoding strategies.
- Advanced Solving Techniques: Implement and evaluate advanced deduction patterns (XWing, Swordfish) within the ASP encoding to handle more challenging instances.
- Extended Irregularity Models: Explore different region generation algorithms and their impact on solving complexity.
- Comparative Analysis: Conduct thorough comparisons with traditional CSP solvers and SATbased approaches on the same dataset.
- Scalability Optimization: Investigate partitioning strategies and heuristic optimizations for very large irregular Sudoku instances ($N > 9$).
- Community Dataset: Expand the generated dataset into a community benchmark for irregular Sudoku solving, inviting contributions from other researchers.

The developed framework provides a solid foundation for advancing constraint solving in irregular grid-based puzzles, with ASP proving to be a powerful and expressive tool for this domain.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] Martin Gebser, Benjamin Kaufmann, Andre Neumann, and Torsten Schaub. 2007. clasp: A conflict-driven answer set solver. In International Conference on Logic Programming and Nonmonotonic Reasoning. Springer, 260-265.

[2] Martin Gebser, Benjamin Kaufmann, Andre Neumann, and Torsten Schaub. 2012. Conflict-

[3] driven answer set solving From theory to practice. Artificial Intelligence 187 (2012), 52–89.

[4] Agnes M Herzberg and M Ram Murty. 2007. Sudoku squares and chromatic polynomials. Notices of the AMS 54, 6 (2007), 708–717.

[5] Donald E Knuth. 2000. Dancing links. arXiv preprint cs/0011047 (2000).

[6] Alan K Mackworth. 1977. Consistency in networks of relations. Artificial intelligence 8, 1 (1977), 99–118.

[7] Takayuki Yato and Takahiro Seta. 2003. Complexity and completeness of finding another solution and its application to puzzles. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 86, 5 (2003), 1052–1060.Publishers.