AI-Powered Sudoku Solver using Search Algorithms

Siya Brahmbhatt (svb52@scarletmail.rutgers.edu, Designer and Guide),

Paige Brathwaite (pb622@scarletmail.rutgers.edu, Principal Implementer of the Prototype App)

Raashi Maheshwari (rm1622@scarletmail.rutgers.edu, Principal Designer of Evaluation Study)

Department of Computer Science, Rutgers University

Professor Kulikowski

CS440 - Introduction to Artificial Intelligence

May 13th, 2025

Table of Contents

I. Introduction

Artificial Intelligence (AI) has revolutionized the way we solve structured problems, particularly in domains governed by clear rules and constraints. One such domain is puzzle-solving, which offers a practical gateway into the implementation of AI algorithms and heuristic methods. Sudoku, a logic-based number placement puzzle, is a classic example of a constraint satisfaction problem (CSP). Sudoku's requirement that each digit from one to nine must appear exactly once in every row, column, and subgrid makes it an ideal candidate for exploring AI search algorithms and heuristic strategies.

Our interest in this project emerged from our daily shared routines involving puzzle games like Sudoku and Wordle, commonly featured in publications such as *The New York Times.* Additionally, Sudoku's visual and logical framework resembled a previous CS440 assignment involving grid-based backtracking for pathfinding, making it both familiar and academically relevant to the course. Motivated by these experiences, we aimed to deepen our understanding of AI concepts by implementing a Sudoku solver that could contrast the effectiveness of uninformed and informed search algorithms.

This project served two main objectives. First, we sought to compare the performance and scalability of a basic depth-first backtracking with that of A* search guided by the Minimum Remaining Value (MRV) heuristic. Second, we aimed to create a modular Python tool that could animate the solving process via the command line, accept arbitrary puzzle inputs, and output diagnostic metrics such as runtime, recursion depth, node expansion, and memory footprint. These goals not only reinforced theoretical AI concepts but also demonstrated how heuristic design can dramatically enhance search performance in real-world CSPs.

ll. Rationale and Goals

Sudoku Puzzles present a well-structured challenge rooted in logic and constraints, making them a strong fit for AI-based problem solving. Our rationale for selecting this project stems from its alignment with core topics in CS440, such as constraint satisfaction, search algorithms, and heuristic design. The goal was to implement two solvers, one uninformed with backtracking and one informed (A* with MRV), to help demonstrate the computational advantages of heuristics and test their scalability across puzzles of varying complexity. We also aimed to build a user-friendly interface that could support the visualization and analysis of performance metrics. See figure 4.

lll. Materials and Procedures

Prior Work and Conceptual Foundation

To inform our design, we explored a range of external studies, tutorials, and thought pieces that documented diverse AI strategies for solving Sudoku. One foundational source was David Carmel's article "*Solving Sudoku by Heuristic Search"* (2021), which laid out how A* search, paired with intelligent heuristics like Minimum Remaining Value (MRV), can significantly reduce unnecessary state expansions by mimicking human prioritization in problem-solving. Carmel explains how heuristic design can influence the branching factor in recursive algorithms, providing valuable lessons in tuning performance through informed search (Carmel).

Another pivotal resource was the SERP AI article, "AI-Powered Sudoku Solving: From Algorithms to Strategy," which offered a comprehensive overview of how different AI paradigms – such as constraint propagation, backtracking, and machine learning – have been applied to structured logic puzzles like Sudoku. This article was particularly useful in comparing

traditional AI approaches with more experimental neural network-based solvers, helping us narrow our focus toward CSP-based strategies that were more aligned with our course content (SERP AI).

We also drew insights from research conducted by the KU Leuven Artificial Intelligence group, specifically their 2023 post "*Sudoku Assistant – An AI Assistant Combining Machine Learning and Reasoning."* This study demonstrated how machine reasoning, when combined with learning from prior solutions, could emulate human logic pathways to progressively refine the solving process. Their feedback loop model – where each solved puzzle informs the handling of future cases – highlighted the potential of hybrid systems that mix symbolic reasoning with learned behavior (Berden and Guns). Together, these sources provided us with both theoretical grounding and practical insights that directly shaped the architecture of our solver and informed our choice to prioritize heuristic-driven A* search for scalability and performance.

Team Roles and Responsibilities

- **Paige Brathwaite**: Developed core functionality for both search algorithms, optimized recursion, and managed performance tracking scripts.
- **Raashi Maheshwari**: Led the benchmarking framework, analyzed experimental results, and created visualizations to communicate findings.
- **Siya Brahmbhatt**: Directed the overall timeline, managed group collaboration, and compiled documentation and final deliverables.

Implementation Process and Technical Setup

The Sudoku solver was implemented in Python using a modular file structure:

- main.py managed control flow and execution timing.
- solver/ contained the backtracking and A* logic.

- heuristics.py implemented the MRV heuristic function.
- utils.py handled input parsing and board modeling.
- data/ stored test puzzles in a standardized format.

While implementing the solvers, we faced several challenges. The backtracking algorithm was initially prone to deep recursion and inefficient branching. We refined this by improving constraint checking and eliminating redundant state evaluations. Meanwhile, tuning the MRV heuristic for balanced performance across easy and hard puzzles required significant experimentation. Although we initially planned to implement a GUI, time constraints led us to prioritize command-line visualization, which proved invaluable for debugging and real-time insight into the solving process.

IV. Results

Our results compare two AI algorithms: Backtracking and A* Search with the Minimum Remaining Value (MRV) heuristic, for solving Sudoku puzzles treated as constraint satisfaction problems. Both approaches were evaluated based on correctness, efficiency(runtime), and scalability.

Correctness: Both algorithms successfully solved all test cases in our dataset. Each solution adhered to Sudoku rules, validating the implementations. The programming uses the is_valid () function to enforce the Sudoku constraints based on rows, columns, and subgrids.

Efficiency: Each solver was timed across puzzles of varying difficulty. The backtracking algorithm performed well on easy puzzles, however slowed on puzzles with limited information. The A* Search algorithm outperformed backtracking in all puzzles by using the MRV heuristic to prioritize cells with the fewest legal options. The first table compares the same (easy) puzzle

with speed. The second and third tables compare two puzzles out of 7 that we tested. The figures compare 4 different puzzles as well as 2 of the same ones. See Figures 1-3.

Heuristic Impact: The MRV heuristic allowed the A* to be enhanced. Using this, the most constrained variable at each step was chosen, and A* drastically reduced the branching factor and avoided unproductive search paths. The heuristic allowed the solver to focus on the cells with the most urgency.

Scalability: As the puzzles increased in difficulty, the difference between backtracking and A* grew in significance. Although backtracking guarantees a solution, it becomes expensive. The A*, with MRV, maintains feasibility even with fewer initial clues.

V. Conclusions from Results

This project successfully demonstrates the power of AI in solving constraint satisfaction problems through the implementation and comparison of Backtracking and A* Search algorithms.

Key takeaways include:

1. AI Efficiency Improves with Domain Knowledge: Heuristics like MRV significantly enhance performance by reducing wasted search effort.
2. Backtracking is Simple yet Expensive: Although effective for basic puzzles, backtracking quickly becomes impractical without pruning or optimization.
3. Human-Like Approach: The A* algorithm mimics the natural way humans solve Sudoku by focusing first on the most constrained or critical cells.

4. Concepts in Practice: Concepts like search space traversal, constraint propagation, and node prioritization have real-world applications in resource scheduling, planning, and decision-making.

If we were to continue the project, we would:

- Explore additional heuristics like Least Constraining Value (LCV)
- Implement forward checking to improve pruning.
- Add a graphical user interface to allow users to input and visualize puzzles interactively.

VI. Impact and Significance of Results

The core AI methods that powered our Sudoku solver were search algorithms (Backtracking and A*) combined with constraint satisfaction heuristics (Minimum-Remaining-Value heuristic and forward-checking constraint propagation). Specifically, the uninformed search method (Backtracking) systematically explores the solution space in a depth-first manner, offering a baseline for us to evaluate a minimum performance. On the other hand, the informed search method (A*), guided by our MRV heuristic, selects the most constrained cell first (or rather those with the fewest valid digit options) therefore pruning the search tree to overall speed up the solution process. Additionally, we also used forward-checking, which allowed us to immediately identify and eliminate inconsistent assignments early in the search. These classic AI techniques helped us to highlight the core principles of heuristic-driven search, and clearly demonstrate how choosing and combining specific AI methods can drastically enhance computational efficiency and solution feasibility.

The performance improvements we demonstrated when comparing A* Search guided by the MRV heuristic against backtracking emphasizes a critical principle in artificial intelligence;

that heuristic choice dramatically shapes performance. Our solver, by effectively using constraint propagation through heuristic-driven node selection, reduced our computational work significantly. This illustrates that smart heuristics don't only speed up search but fundamentally transform lengthy problems into manageable ones. Practically, these results emphasize the effectiveness of classical AI techniques for solving structured constraint-satisfaction problems, highlighting their potential application beyond academic contexts. A well-optimized solver like ours could efficiently power real-world applications, including educational tools, automated puzzle-generation frameworks, or accessibility-focused puzzle aids. Moreover, the demonstrated scalability and modularity of our implementation provide a strong foundation for extending our solver into more complex puzzles and variants, such as larger Sudoku boards or other logic puzzles, making the implications of our results broadly relevant. Ultimately, our findings reinforce the central message in AI research which is that smart heuristic design combined with robust algorithmic structure can completely transform a search process.

VII. Learnings as a Group and Alternative Approaches

We first noticed that the heuristic selection was the single biggest factor on search performance. Measuring backtracking against A* + MRV showed speed-ups times that jumped from single-digit percentages improvements on easy puzzles to multiple orders of magnitude on expert puzzles. That taught us that algorithm and heuristic choices matter more than low-level optimizations: a well-chosen heuristic can eclipse hours of code-level micro-tuning (which is the major difference as to why adaptive algorithms are more efficient).

Next, we had found that splitting the project into different classes such as utils.py (I/O + board model), heuristics.py, and other solver classes meant any team-member could replace a

module without ripple effects. For example, swapping the priority-queue implementation in astar_solver.py took minutes because our code was clear and modular. This swapability design allowed for team members to be able to make more of their own edits to the code without worrying too much about having to also finetune other aspects.

Finally, we learned that using command-line visualisation is an under-rated debugging tool. Our “live board” display within the terminal let us *see* the recursion unfold one row at a time which surfaced logic bugs (our A* solving correctly but not displaying correctly) long before we had started our debugging tests. Additionally, because we kept the output terse (overwriting a single board instead of printing every branch) we avoided the “scroll-wall” that typically makes recursive debugging painful. We will definitely carry this trick into future projects that involve deep search trees.

In regards to changes we would make in the future, we would definitely manage our data differently. We underestimated the amount of data overhead that we would have to manage, and the amount of testing we would have to do along with that. Embedding an automated data-collection pipeline into the codebase instead of gluing together impromptu scripts near the deadline would have saved us a headache. By wiring logging hooks directly into every solver call, we could have streamed run-time, node expansion counts, and memory statistics straight into tidy CSV or Markdown tables. Bookkeeping that information ourselves stopped us from monitoring the regressions continuously and focusing our energy on experimentation.

A second improvement would be to introduce larger board sizes, such as 16 × 16 Samurai puzzles, much earlier in development. We had toyed with the idea of testing boards larger than 9x9 to have variation within the solver but had decided against it because of time-constraint

reasons. With more time, we definitely would have chosen to add it in as another feature of the project instead of only having the simplest board. As it did keep our difficulty level manageable, doing that would have also stress-tested our methods and exposed scalability bottlenecks sooner.

Finally, we would formalise pair-programming and group code reviews as a routine rather than doing so when we ran into issues. The few sessions we did schedule surfaced subtle off-by-one errors and display glitches far faster than solo debugging ever did. Setting a weekly meeting for collaborative walk-throughs would highlight that benefit, ensuring fresh eyes that are regularly sweeping the codebase and allowing for shared design knowledge to grow evenly across the team.

Taken together, these adjustments would make the next iteration of this project not only more robust but also far smoother to manage in regards to scaling the project in size and ambition.

X. Individual Statement

As the Principal Designer of the Evaluation Study, my role in this group project centered around quantifying the performance and effectiveness of our AI-based Sudoku solver through rigorous experimental analysis. I was responsible for designing and implementing the benchmarking framework that allowed us to evaluate the solver's performance across puzzles of varying difficulty levels. This included collecting data on runtime, node expansion, and memory usage, and transforming that data into clear visualizations that enabled informed decision-making during development.

I also contributed to the validation of both the uninformed (Backtracking) and informed (A* + MRV) solvers by developing comparison tables, stress tests, and experiment logs that we later used to interpret trends and performance gaps. My approach focused on translating raw technical output into understandable and actionable metrics. This was critical in refining our heuristic design and optimizing our code. Additionally, I helped ensure fairness in experimental design by standardizing input puzzles and logging methods, which made performance evaluation across solvers both accurate and reproducible.

From a collaborative standpoint, I played a key role in cross-verifying results with my teammates, assisting in debugging when performance anomalies arose, and communicating insights during team syncs. I also contributed to portions of the final report that involved results interpretation and impact discussion, ensuring that our conclusions were evidence-based and grounded in both quantitative results and theoretical understanding.

Through this group project, I learned how to approach evaluation as an integral part of software design, rather than as an afterthought. I significantly improved my ability to work with

runtime data, design reproducible experiments, and synthesize results into narratives that inform both technical improvements and broader insights. I also came to appreciate the power of heuristics—how choices like MRV not only enhance solver efficiency but also simulate a level of human logic in algorithmic design. These lessons have reinforced my interest in applying AI evaluation methods to real-world domains, particularly in areas like cybersecurity and machine learning, where performance benchmarking is crucial.

Overall, this experience has sharpened both my technical and analytical thinking, while deepening my appreciation for collaborative project design in AI. It has equipped me with skills that I am excited to carry forward into future academic research and professional work in data science and artificial intelligence.

VIII. References

"AI-Powered Sudoku Solving: From Algorithms to Strategy." SERP AI, https://serp.ai/posts/game-of-sudoku/. Accessed 9 May 2025.

Berden, Senne, and Tias Guns. *"Sudoku Assistant – An AI Assistant Combining Machine Learning and Reasoning."* AI @ KU Leuven, 9 Feb. 2023, https://ai.kuleuven.be/stories/post/2023-02-08-sudoku. Accessed 9 May 2025.

Carmel, David. *"Solving Sudoku by Heuristic Search."* *Medium*, 17 Sept. 2021, https://medium.com/@davidcarmel/solving-sudoku-by-heuristic-search-b0c2b2c5346e. Accessed 9 May 2025.

Quach, Katyanna. "AI Taught to Beat Sudoku Puzzles. Now How about a Time Machine to 2005?" *The Register*, 1 Dec. 2017,

https://www.theregister.com/2017/12/01/ai_machine_leaning_sudoku/. Accessed 9 May 2025.

Carmel, David. “Solving Sudoku by Heuristic Search.” *Medium*, 17 Sept. 2021, https://medium.com/@davidcarmel/solving-sudoku-by-heuristic-search-b0c2b2c5346e. Accessed 9 May 2025.

Berden, Senne, and Tias Guns. “Sudoku Assistant – An AI Assistant Combining Machine Learning and Reasoning.” *AI @ KU Leuven*, 9 Feb. 2023, https://ai.kuleuven.be/stories/post/2023-02-08-sudoku/. Accessed 9 May 2025.

IX. Appendix (with Tables and Figures)

Solver	Time(s)
A* (MRV)(many clues)	0.2025
Backtracking(many clues)	0.2455

Figure 1

Puzzle	Solver	Time (s)
Moderate Clues(#1)	A* (MRV)	0.2215
Few Initial Clues(#2)	Backtracking	0.4228

Figure 2

Puzzle	Solver	Time (s)
Few Initial Clues(#5)	A* (MRV)	0.5215
Few Initial Clues(#4)	Backtracking	0.6728

Figure 3

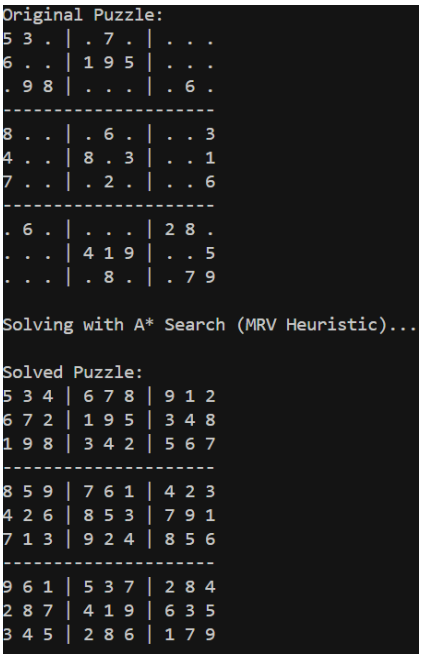
Original Puzzle:
5 3 . | . 7 . | . . .
6 . . | 1 9 5 | . . .
. 9 8 | . . . | . 6 .
8 . . | . 6 . | . . 3
4 . . | 8 . 3 | . . 1
7 . . | . 2 . | . . 6
. 6 . | . . . | 2 8 .
. . . | 4 1 9 | . . 5
. . . | . 8 . | . 7 9
Solving with A* Search (MRV Heuristic)...
Solved Puzzle:
5 3 4 | 6 7 8 | 9 1 2
6 7 2 | 1 9 5 | 3 4 8
1 9 8 | 3 4 2 | 5 6 7
8 5 9 | 7 6 1 | 4 2 3
4 2 6 | 8 5 3 | 7 9 1
7 1 3 | 9 2 4 | 8 5 6
9 6 1 | 5 3 7 | 2 8 4
2 8 7 | 4 1 9 | 6 3 5
3 4 5 | 2 8 6 | 1 7 9

Figure 4