FINAL PROJECT REPORT

AI Powered One-Up Puzzle Solver

BY:

DHARANIDHAR MANNE

CHOLAYASWANTH KUMAR GOLLA

RAVI PRASAD GRANDHI

SRAVYA REDDY KAITHA

JITENDRA KADIYAM

TABLE OF CONTENTS

ABSTRACT

This project addresses the challenge of solving the "Oneupuzzle" game, a logic-based puzzle played on an (n times n) grid. Each grid contains vertical and horizontal walls, as well as blocked cells, creating smaller, segmented regions within the board. The objective is to assign numerical values to each valid cell within the constraints imposed by the walls and blocks while ensuring each segment adheres to specific rules.

The solution employs a backtracking approach, combined with forward propagation, to iteratively narrow down possible values for each cell and verify the feasibility of assignments. This approach effectively reduces the solution space and ensures optimal performance. The algorithm integrates domain-specific analysis for rows and columns, leveraging constraints to enhance computational efficiency.

The implementation further includes a visualization tool to render the puzzle grid and solution. This tool highlights blocks, walls, and the values assigned to cells, aiding in result interpretation and analysis. By combining robust algorithms with intuitive visual outputs, this project demonstrates a practical application of computational techniques in solving complex logical puzzles.

This project contributes to the broader understanding of constraint satisfaction problems, showcasing their applicability in both recreational and real-world scenarios.

INTRODUCTION

Puzzle-solving has always intrigued individuals across different domains due to its combination of logic, creativity, and critical thinking. Among the many types of puzzles, the "Oneupuzzle" stands out as a distinctive problem involving constraints and spatial logic. The Oneupuzzle operates on an (n times n) grid, with walls and blocks dividing the grid into smaller segments. Each segment must satisfy specific conditions, making the puzzle both challenging and rewarding to solve. This project explores the methodology for solving the Oneupuzzle using computational techniques, emphasizing the integration of artificial intelligence principles to address its inherent complexity.

The Oneupuzzle belongs to the family of constraint satisfaction problems (CSPs), which are mathematical problems defined by a set of objects and constraints that the objects must meet. These problems are prevalent in various applications, from scheduling and resource allocation to logic-based games. The essence of solving a Oneupuzzle lies in systematically assigning numerical values to cells in a way that respects the constraints imposed by the

walls, blocks, and grid boundaries. This complexity makes it a compelling case study for the application of backtracking and propagation techniques

Backtracking, a well-established problem-solving technique in artificial intelligence, serves as the cornerstone of this project. It explores all possible configurations of the puzzle by recursively attempting to assign values to cells and retracting assignments when conflicts arise. To improve efficiency, the project employs forward propagation, a technique that proactively reduces the solution space by eliminating infeasible options before making further assignments. Together, these techniques ensure a methodical exploration of the puzzle space, balancing computational cost and accuracy.

The project also incorporates domain-specific analysis to optimize the puzzle-solving process. By dissecting the grid into rows, columns, and smaller segments, the algorithm identifies localized constraints that narrow down possible values for each cell. This segmentation not only reduces the computational complexity but also aligns with the intuitive nature of puzzle-solving, where smaller problems are tackled independently before assembling the larger solution.

Another significant aspect of the project is its focus on visualization. A robust visualization tool has been developed to illustrate the puzzle grid, including the placement of walls, blocks, and the final solution. Visualization plays a crucial role in enhancing user understanding, providing a clear representation of how the solution adheres to the puzzle's constraints. It also aids in verifying the correctness of the algorithm, offering insights into the decision-making process during puzzle resolution.

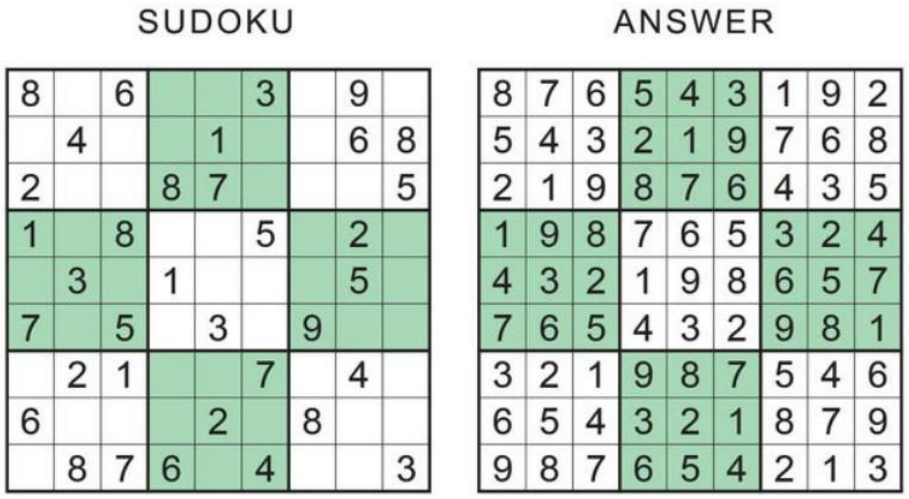
This project holds relevance beyond recreational puzzle-solving. Constraint satisfaction problems, like the Oneupuzzle, have applications in numerous real-world scenarios, such as scheduling tasks, planning routes, or designing optimal layouts. The methodologies demonstrated here can be extended to tackle these challenges, showcasing the versatility and power of computational approaches in addressing structured, constraint-driven problems.

Through the exploration of the Oneupuzzle, this project contributes to the broader understanding of algorithmic problem-solving. It highlights the interplay between logical reasoning and computational efficiency, emphasizing the importance of combining established techniques with innovative strategies to solve complex puzzles. By doing so, it underscores the potential of algorithms like backtracking and propagation in solving a wide range of constraint-based challenges.

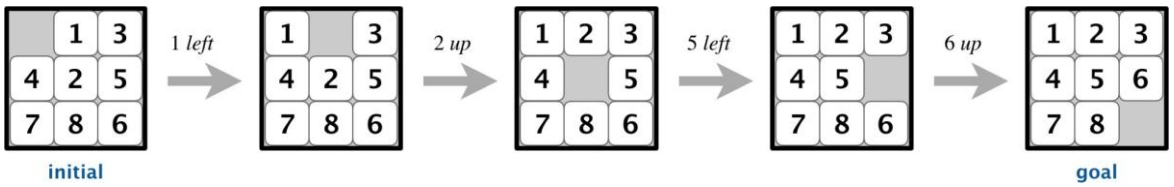
POPULAR PUZZLE GAMES

Puzzle games have captivated audiences with their combination of logic, strategy, and creativity. Here are five popular puzzles and the algorithms commonly used to solve them:

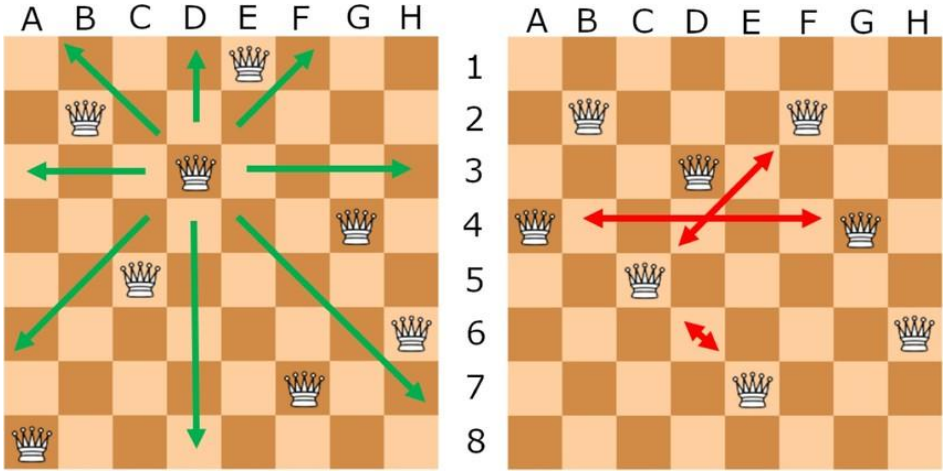
1. **Sudoku:** A classic number-placement game on a 9×9 grid, where numbers 1–9 must appear exactly once in each row, column, and 3×3 subgrid. Sudoku is often solved using backtracking, a trial-and-error algorithm that systematically fills numbers and backtracks when conflicts arise. Advanced techniques like constraint satisfaction programming (CSP) or Knuth's Algorithm X.



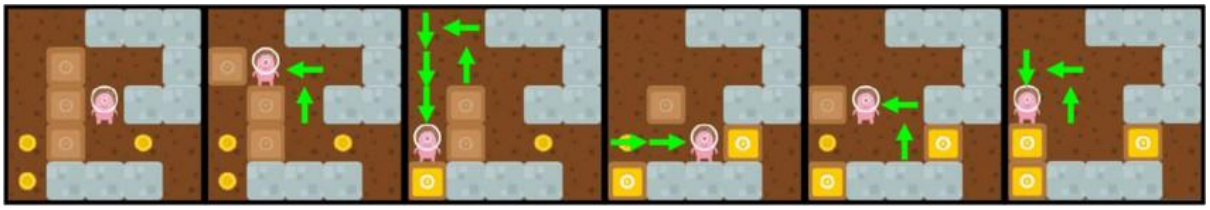
2. **8-Puzzle Problem:** A sliding tile puzzle where tiles must be arranged sequentially in a 3×3 grid. Solving it involves heuristic-based search algorithms such as A*, which uses Manhattan distance to estimate the steps needed to reach the goal. Breadth-First Search (BFS) and Depth-First Search (DFS) are also used but are less efficient for optimal solutions.



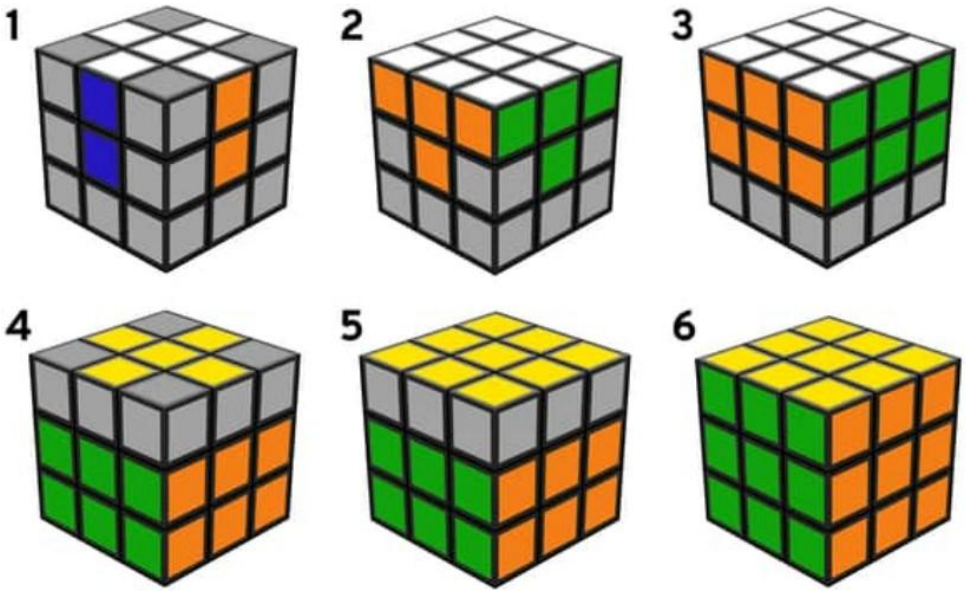
3. **Eight Queens Puzzle:** This chess-based puzzle involves placing eight queens on a chessboard such that no two queens threaten each other. Solvers employ backtracking and more advanced methods like genetic algorithms or minimum conflict algorithms for larger variants.



4. **Sokoban:** A box-pushing puzzle where boxes must be moved to designated storage locations without getting stuck. Algorithms like A*, using heuristics to minimize box movements, and Zobrist hashing with iterative deepening are effective in solving complex layouts.



5. **Rubik's Cube:** The 3D combination puzzle requires aligning colors on each face. While manual solving employs layer-by-layer methods, algorithms like Kociemba's Two-Phase Algorithm are used computationally to solve the cube in the fewest moves by reducing complexity at each phase.



These puzzles not only entertain but also provide benchmarks for testing algorithms in artificial intelligence and optimization, illustrating the intersection of logic, creativity, and computational efficiency.

METHODOLOGY

1. Dataset Collection and Encoding

For this project, five sample Oneupuzzle grids were collected from the official Oneupuzzle website. These grids varied in dimensions, including 4×4, 5×5, 6×6, and 7×7 layouts. Each grid introduces a mix of empty cells, blocks, and walls, adding complexity to the puzzles. To facilitate computational processing, the grids are

encoded into two distinct string representations: ‘gamestr_h’ for horizontal details and ‘gamestr_v’ for vertical details.

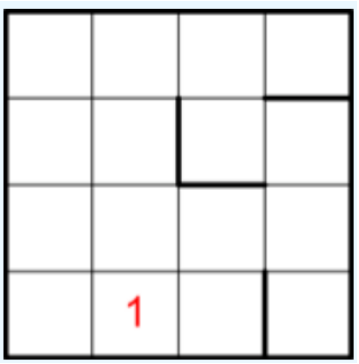
1. Horizontal String (‘gamestr_h’):
    - Each row’s end is marked with ‘r’.
    - Empty cells are represented as ‘x’.
    - Walls are denoted by ‘w’, while blocks use ‘b’.
    - Non-empty cells containing numeric values directly store their assigned numbers.

2. Vertical String (‘gamestr_v’):
    - Column traversal ends are indicated by ‘c’.
    - Similar to ‘gamestr_h’, walls are marked with ‘w’, blocks with ‘b’, and empty cells with ‘x’. Numeric values are stored as they appear in the grid.

This encoding method ensures the grid's structure and constraints are accurately preserved while maintaining a lightweight representation for computational tasks. For preprocessing and analysis, string representations are parsed to extract grid data, allowing operations such as domain computation, constraint propagation, and visualization.

Such structured encoding is essential for handling grid-based puzzles as it translates visual grid information into a format suitable for algorithmic manipulation, simplifying operations like backtracking or constraint satisfaction.

Input



Output

gamestr_h = "xxxxrxxwxxrxxxxrx1xwx"

gamestr_v = "xxxxcxxx1cxxwxxcxwxxx"

2. Data Conversion into Numerical Format

The process of converting encoded string representations of a Oneupuzzle into numerical formats is critical for computational analysis and problem-solving. The project achieves this through systematic parsing, segmentation, and domain assignment.

- Parsing the Grid : It processes the `gamestr_h` string to interpret the horizontal features of the grid. This includes identifying rows (‘r’), walls (‘w’), blocks (‘b’), empty cells (‘x’), and assigned numerical values. Each cell's attributes are stored in a structured format within the board. Similarly, it processes `gamestr_v`, handling columns (‘c’), alongside walls, blocks, and cell values. Together, these functions translate the grid's textual representation into an initial structured data model.

Input

gamestr_h =

"xx23bxxrxxxxxx6rxxxxwxxxrxxxxxxwxrxxxx6xxr2xxxxxxrxxxwxxxx"

gamestr_v =

"xxxxx2xcxxxxxxxc2xxxwxwxxc3xxwxxxxcbxxx6xxcxxxxwxxxcx6xxxxx"

Output

```
x x 2 3 b|x x
x x x x x x 6
x x x x|x x x
       -
x x x x x x|x
     -     -
x x x x 6 x x
     -
2 x x x x x x
x x x|x x x x
```

- Merging Segments : After parsing, it integrates overlapping horizontal and vertical segments. This step ensures that the grid structure is cohesive, with each segment uniquely defined. For instance, two adjacent walls that form a continuous barrier are treated as a single segment. This reduces redundancy and streamlines subsequent computations.

- Segment Analysis : It analyzes rows and columns independently. It counts the number of valid cells in each segment, enabling the computation of feasible domains for numerical assignment. By segmenting the grid, the problem is broken into smaller, manageable subproblems, aligning with the constraints imposed by walls and blocks.

- Domain Assignment : The final step involves assigning numerical domains to each cell based on the parsed and segmented data. For cells marked as empty ('x'), the domain includes all valid values constrained by the segment size. This ensures that solutions adhere to the rules of the Oneupuzzle, such as maintaining unique values within a segment. Cells with predefined values directly retain their respective domains.

This structured conversion transforms the grid from a human-readable format into a computationally efficient numerical representation. It enables the application of backtracking and constraint satisfaction algorithms, streamlining the process of solving complex puzzles. Advanced encoding techniques, like those discussed in categorical data processing, ensure the integrity and interpretability of numerical data during algorithmic analysis.

3. How an Algorithm Solves the Problem

It uses a combination of backtracking and forward propagation to solve the Oneupuzzle by systematically assigning numerical values to grid cells while adhering to the puzzle's constraints. This approach explores potential solutions recursively, ensuring that all constraints are met before finalizing the puzzle's state. Here's a step-by-step explanation of how the algorithm works:

- Initial Setup : The function begins by ensuring that the grid is pre-processed into a structured format. This includes assigning potential values (domains) to each cell based on the grid's dimensions and constraints. Cells that are empty ('x') are assigned a domain of valid numbers, while cells with predefined values retain those numbers as their sole domain.

- Forward Propagation : Forward propagation reduces the solution space by eliminating invalid options. If a cell has a single valid value, this value is removed from the domains of its neighbors (cells in the same row, column, or segment). This step iteratively simplifies the grid, reducing the need for exhaustive exploration later in the process.

- Backtracking : Once forward propagation cannot further simplify the puzzle, the backtracking algorithm is invoked. This recursive process attempts to assign a value from the cell's domain, starting with the most constrained cell (i.e., the one with the smallest domain). After assigning a value:

A. The algorithm validates the assignment by checking its consistency with the grid's constraints.
B. If valid, the algorithm recursively proceeds to the next cell.
C. If invalid or if the recursive step fails to find a solution, the algorithm backtracks by removing the assignment and trying the next possible value in the domain.

- Termination : The algorithm terminates successfully when all cells are assigned valid values that satisfy the constraints. If no solution exists, it concludes that the puzzle is unsolvable.

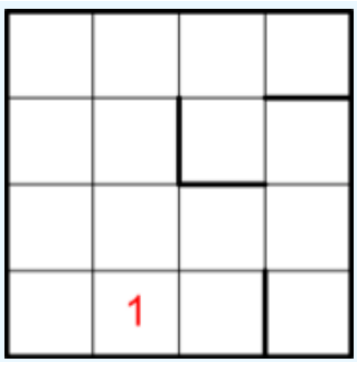
Efficiency Considerations

- Heuristic Selection: The algorithm leverages heuristics to prioritize cells with smaller domains, significantly reducing unnecessary computation.
- Constraint Propagation: The forward propagation step ensures that the algorithm works efficiently, even for larger grids, by narrowing down possibilities before backtracking.
- Recursive Depth Limiting: Backtracking depth is managed to prevent infinite recursion in cases of unsolvable configurations.

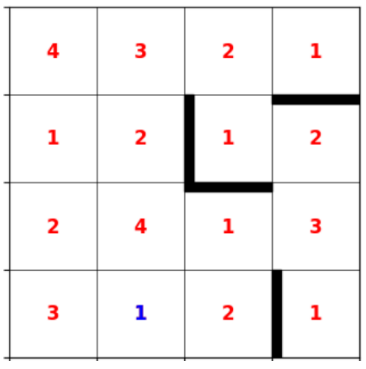
4. Result and Visualization

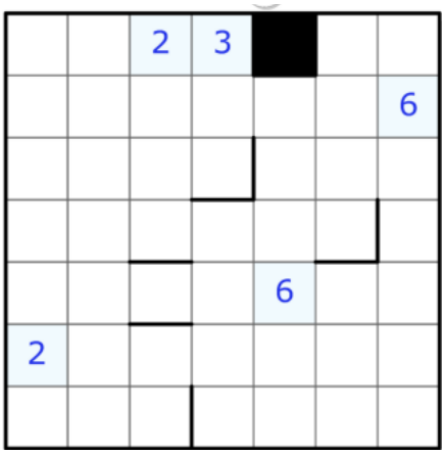
Generates a grid representation where each cell is colored based on its value, walls ('w') and blocks ('b') are marked clearly, and the solution's assigned numbers are displayed within the grid. Blue and red colors are applied to numbers for better differentiation, making it easy to understand the puzzle's structure and the final solved state. Visualization enhances the clarity of the solution, providing an intuitive understanding of the grid's transformation throughout the solving process.

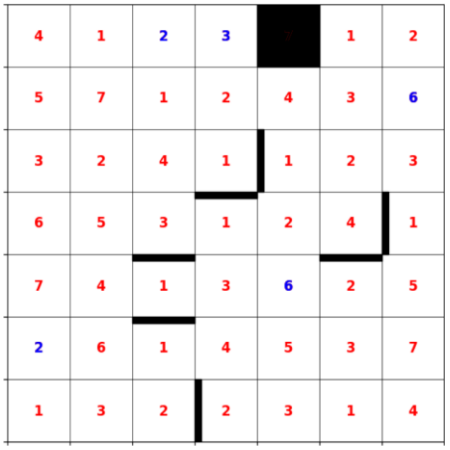
Initial State



Solution







CHALLENGES

One of the primary challenges encountered in solving the Oneupuzzle lies in the complexity of the grid itself. As the grid size increases (e.g., from 4×4 to 7×7), the solution space grows exponentially, making the backtracking algorithm computationally intensive. This complexity is further exacerbated by the presence of walls and blocks, which divide the grid into smaller segments, requiring careful domain management for each cell. The solution's efficiency depends on effectively handling constraints and reducing the search space through forward propagation. Additionally, ensuring accuracy during the conversion of the puzzle grid from a string representation to a numerical format can be tricky, especially when parsing complex grid layouts.

Another challenge involves handling the diversity of Oneupuzzle instances with varying grid sizes and segment configurations. Optimizing the algorithm to handle such variability, while ensuring consistent performance, is an ongoing task. Visualizing the solution efficiently also becomes harder as the grid size increases, necessitating more advanced graphical techniques for rendering.

FUTURE WORK

A significant enhancement for future iterations of this project would involve incorporating OpenCV and an OCR (Optical Character Recognition) engine to directly process images of the Oneupuzzle grid and convert them into the string representation used by the algorithm. This would eliminate the need for manual input and streamline the puzzle-solving process. OpenCV could be employed to detect grid boundaries, walls, blocks, and other essential elements in the image, while OCR would extract numerical values from each cell.

This development would also allow the system to handle images of varying quality and complexity, enabling real-time solving of puzzles directly from photographs or screenshots. Additionally, this could pave the way for a more interactive user experience, where users could input their puzzle images directly, and the system would process and solve them automatically. Improving the accuracy and speed of OCR processing would be essential to handle the dynamic nature of grid layouts and ensure a seamless transition from image to algorithm.

CONCLUSION

This project demonstrates the effectiveness of backtracking and constraint propagation for solving Oneupuzzle. The integration of visualization techniques aids in interpreting the solution. Future enhancements, such as OpenCV and OCR integration, could further streamline puzzle-solving, enabling users to easily convert images to solvable grid representations.

REFERENCES

1. Simonis, H. (2005). Sudoku as a constraint problem. In Proceedings of the CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems.

2. Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search.

3. Knuth, D. E. (2000). Dancing links. Communications of the ACM, 46(8), 96-101.

4. Haralick, R. M., & Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems.

5. Russell, S., & Norvig, P. (2020). Constraint satisfaction problems. In Artificial Intelligence: A Modern Approach (4th ed., pp. 198-232). Pearson.

6. Van Hentenryck, P., & Saraswat, V. (1997). Constraint programming for combinatorial optimization: A tutorial. AI Magazine, 18(4), 32-44.