CSCE A405 - Artificial Intelligence
Assignment 4

Constraint Satisfaction

Solving 9×9 Sudoku using constraint satisfaction algorithms.

Jon Rippe jfrippe@alaska.edu

KivaGale Hall UAA@KivaGale.com

Overview

The traditional Sudoku puzzle consists of a 9×9 grid with various grid spaces pre-filled with a value 1-9. These spaces are pre-filled such that there is only one possible solution to the Sudoku puzzle. A solution is found when all spaces have been filled with a value 1-9 with no value repeating itself in any given row, column or 3×3 block (see Figure 1).

Solving a Sudoku puzzle is done by analyzing the constraints on each space - the values that *could* be placed in a space without creating a conflict with another space's set value - and setting a space's value once its constraints have been narrowed down to a single possible value. Our Sudoku solving AI utilizes the three algorithms described below to solve any Sudoku puzzle. Additionally, the AI is capable of solving puzzles of different sizes (e.g., 16×16, 25×25, etc.), however larger puzzles become exponentially more complex and our AI has not been optimized to efficiently handle puzzles of such complexity. Two sample 16×16 puzzles are included in the demo program.

Figure 1: Sudoku puzzle unsolved (top) and solved (bottom).

Constraint Algorithms

1. Naive Constraint Propagation

A simple algorithm. It checks every space that has previously been modified (utilizing a queue) and updates its constraints based on its neighbors' values and its neighbors' constraints based on its value. A neighbor is classified as any space in the same row, column, or block as the given space. If the space's constraints or any of its neighbors' constraints are reduced to length 1, set the respective space's value to its only constraint and clear its constraints. If any changes are made, the affected spaces are put onto the queue.

2. Naked Set Propagation

Finds all naked sets within a group (i.e., row, column, or block). This algorithm retrieves a subset of the powerset of each group. This subset contains all sets within the group that contain 2 or more spaces. The constraints for each space within each set are combined into a set of constraints that corresponds with the set of spaces. If the cardinality of these two sets (spaces and constraints) is the same but lower than the cardinality of the group set, then the set of spaces must be a naked set. For all naked sets found, remove all values within the corresponding set of constraints from the inverse set of spaces (spaces in the same group but not in the naked set). If any changes are made, the affected spaces are put onto the queue.

3. Recursive Backtracking w/ Iterative Widening

This algorithm is only necessary when the above 2 reach a deadend and the program must make a “guess” to continue. The program systematically runs through all empty spaces, picks a constraint from the space’s possible constraints, and creates a new, identical Sudoku board with that one guess made. Then the entire process is repeated on the new board. If the backtracking finds a solution, it is returned to the calling game. If an unsolved or conflicted board is returned, the calling game removes that constraint from the space and continues with the next constraint.

This algorithm functions identically to a depth-first tree search with no depth limit. To reduce the search space size, the algorithm limits the initial branching factor of the tree, starting with spaces that only have 2 constraint values (max branching factor of 2) and increasing if/when no solution is found. Additionally, the algorithm takes advantage of a hash table that tracks previously seen board states to trim duplicates from the search space.

Processing

The program follows a simple workflow:

1. Run the Constraint Propagation algorithm
2. Run the Naked Set Propagation algorithm
3. Repeat 1 & 2 until the puzzle is solved or neither are able to find anything to update.
4. If the puzzle is not solved, make a guess (starting with 50/50 spaces) and restart from 1.

Testing

Method

Three million Sudoku puzzles with difficulty ratings were downloaded from kaggle.com/radcliffe/3-million-sudoku-puzzles-with-ratings. A testing program was created to read in the puzzles, normalize the difficulties to the desired 1-5 star rating, solve each puzzle, and record the results. Because of size limitations, the full test set has not been included. Instead, a truncated, balanced version of the dataset containing 500 puzzles is included for demonstration along with two expert level 16×16 puzzles randomly generated from https://www.sudoku-puzzles-online.com/hexadoku/print-hexadoku.php. Individual results of the demo set can be found in *results-demo.csv*, which is included. Additionally, the demo program can be run at any time on any properly formatted dataset (see *sudoku-trunc.csv* for example).

The program’s performance on each puzzle is measured by how many constraint propagation iterations were needed to find the solution. The number of recursive calls made was not used as a performance metric because it is correlated with the number of constraint propagation iterations.

Results

As stated above, individual results for the entire test set have not been included due to size limitations. To view individual results, either refer to *results-demo.csv*, run the demo program on *sudoku-trunc.csv*, or download the original dataset, prep, and run using the demo program.

Figure 2 shows a summary of the results from the full dataset of three million puzzles. These results show a distinct correlation between puzzle difficulty and the effort put into finding a solution. On average, each difficulty runs approximately twice as many iterations as the difficulty before it. *Note: Figure 2 uses a logarithmic scale.*

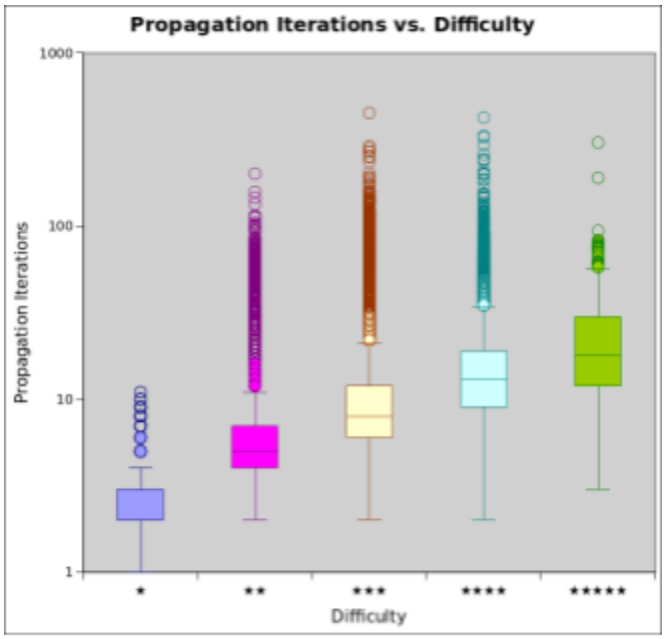

Figure 2: Propagation Iterations vs. Difficulty

It's important to note that there are several outliers that show relatively excessive calculation needed to solve some puzzles. These outliers occur when the program encounters a need to use backtracking early on in the solving process and gets "unlucky" with its initial backtracking selections, which causes the search space to increase exponentially in size. At first glance, the data suggests this happens the most with medium difficulty puzzles, but in reality the patterns seen here are due to the dataset being imbalanced in regard to difficulty classifications (see link to original data).

Conclusion

Although improvements could very likely be made to improve solve time on larger boards, utilizing the three algorithms above our AI is able to solve any 9×9 Sudoku puzzle within a fraction of a second, giving it a 5★ rating in regard to solving traditional 9×9 Sudoku puzzles.