Thomas Varner
Solomon Gaim
Ulrik Soderstrom

Solving Sudoku with Network Analysis

Problem Statement:

Puzzles such as Sudoku are particularly difficult for computers to solved based on the enormity of the solution-spaces involved in these puzzles. The best known algorithm for solving Sudoku: Dancing Links, is based on a guess-and-check approach, although there are known heuristics that can be used to improve the algorithm's runtime. This project aims to create a graphical representation of a Sudoku board that will allow us to use network properties such as degree centrality and flow to implement these heuristics and improve the algorithm's runtime.

Introduction:

Sudoku is a puzzle in which the user is presented with a partially-completed nxn grid of values within the range [1,n], and is tasked to fill in all of the missing values so that each row, column, and section contain no repeated values. For some boards, solving the board is as simple as finding the only values a cell can be, and repeating this process until the board is complete; for other boards, one might attempt this same process until they reach a deadend and cannot proceed without making a guess as to what value a cell on the grid takes. This distinguishes between two classes of sudoku problems: those solvable in polynomial time, and those solvable in nondeterministic polynomial time. The latter type of problem is of great academic interest, for many real-world problems such as protein-folding and movement-planning can be reformulated into this same type of problem. The best known way to solve such problems is to follow a guess-and-check procedure, where the most efficient solvers use some decision process to find the best guess to make at each stage. In this project we formulate a graphical representation of the solution space for a given sudoku board, and use graphical metrics to discuss possible improvements to the most naive guess-and-check methodology for solving nondeterministic puzzles.

Literature Review:

In 1967, Robert Floyd released a paper[3] in which he showed causality between the need to backtrack after making a guess while solving a puzzle, and that puzzle only being solvable in nondeterministic polynomial time. He also alludes to the tree-like nature of the solution space to such problems, where moving "down" the tree using a depth-first search is analogous to making a guess, and moving "up" the tree is analogous to backtracking and removing a guess. With this exhaustive approach towards solving nondeterministic puzzles in mind, Donald Knuth published his Dancing Links[1] algorithm in 1975, presenting an efficient methodology for implementing this type of problem solver, the "obvious trial-and-error approach". 35 years later

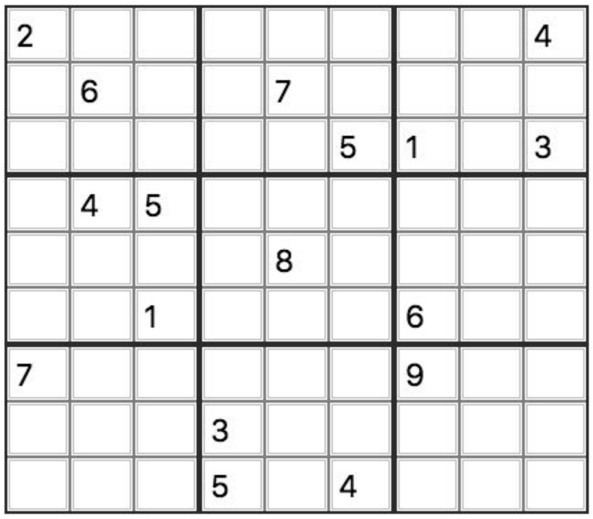
Donald Knuth reflected on his algorithm, and the lack of progress that has been made in solving this sort of problem: “indeed, I can’t think of any other reasonable way to do the job, in general.”

Heuristics however, have been suggested over the years that can improve the efficiency of depth-first, backtracking, exhaustive algorithms; these heuristics generally fall into two main categories. The first category is translative/geometrical exploitations[6] for a class of NP-Complete problems that include solving hypercubes or pentomino puzzles but not solving sudoku boards. The second is making guesses in the solution space that lead to the fewest number of future guesses to be made[4]. This latter heuristic is that we chose to validate using our graphical representations of a sudoku problem.

The number of clues present on a sudoku board has important implications for the type of problem contained on that board as well as its relative difficulty. Gary McGuire showed through exhaustive search that in order for a 9x9 board to have a unique solution, it must have a minimum of 17 clues[5]. This does not mean that a 17-clue board has a unique solution. A minimum number of clues has not been provided for any larger board sizes due to the computational requirements of exhaustive searches on such a scale. For this project we decided to focus on the hardest class of problems for 9x9 boards (~17-20 clues). The number of combined clues and guesses on the board during the progression of the dancing links algorithm also has important implications for the algorithmic progression. When enough values are filled in on the board, the board will enter the polynomial-solvable regime of problems, and no more guesses need to be made for the solver to determine whether its current state is correct or not. The heuristic to the Dancing Links mentioned above is analogous to adjusting the solution tree such that all of the decisions with the most children are at the bottom of the tree.

Methodology:

Data Collection: Sudoku boards were created and collected using a random board generator. This site creates boards with varying difficulty options and as well as initial solutions with only one unique board. [7] 12 of these boards were downloaded and converted to csv files for comparative testing between dancing links and the heuristic developed.



2								4
	6			7				
					5	1		3
	4	5						
				8				
		1				6		
7						9		
			3					
			5		4			

MultiGraph Model: A multi-graph with a node to represent each sub-section, row, column, and cell on the board. Undirected edges connect sub-section, row, and column type nodes to all of the cells they contain. This graph was used to model the game of sudoku, figure out possible values a cell can take, and verify that the complete board is correct. Additionally, this model is generic and works for any sudoku board. Figure 1, below shows a representation of the multigraph.

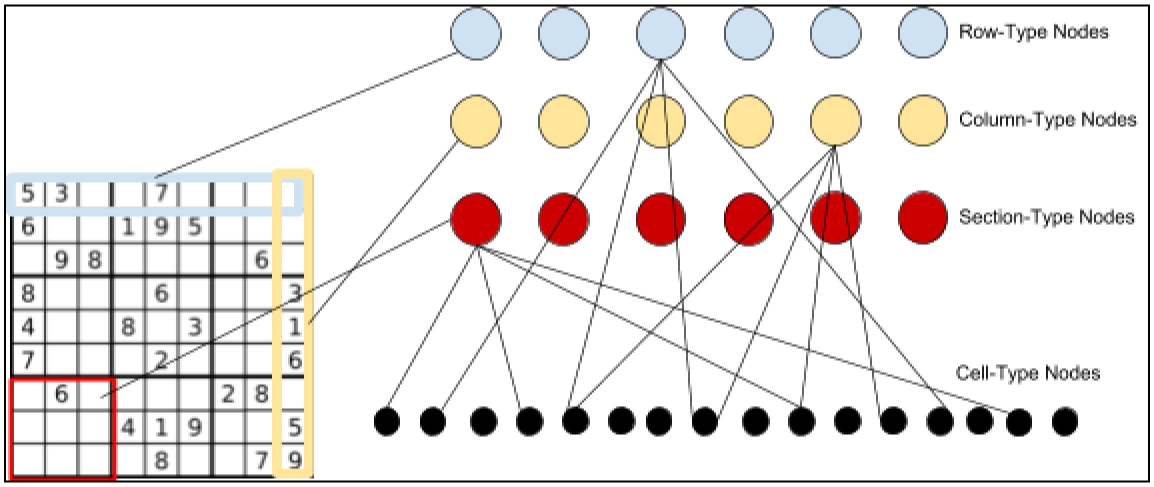


Figure 1: MultiGraph Model

SharedValueModel: A simple undirected graph. Nodes represent cells on the board, undirected edgesconnect nodes that share one or more possible values, and belong to the same row, sub-section or column. This notion of shared values is an indicator for a node’s independance. For example, if a node is connected to many other nodes, guessing a value for that node affects the possible values of each other node it is connected to. Node that components made up of only one node represent cells that contain a clue, and hence have no possible values. Figure 2, below shows a representation of the SharedValueModel.

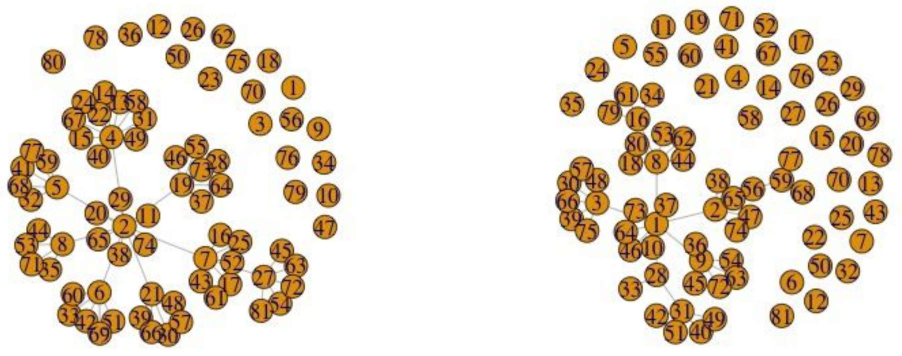


Figure 2: SharedValueModel - Left: Board 1, Right: Board 2.

igraph: The SharedValueModel graph contains a method for converting to a graph in igraph. We used igraph for computation of graph metrics to inform our Dancing Links algorithm with heuristics.

Dancing Links (recursive):

1. Randomly selecting a cell whose value is unknown, and randomly assigning that cell one of its possible values without replacement.
2. Update the possible values each node can take for the entire board and discover any other cell, value pairs that must be true if this first guess is true (find all polynomial-time solutions for the current board state).
    a. If a complete board is found, return True
    b. If the board is found to be invalid, return False
3. Call Dancing Links on current board state
    a. If the next call returns True, return True
    b. If the next call returns False, randomly select a new value from this cell's possible values without replacement and go to (2)
    c. If there are no remaining possible values remaining, return False

Dancing Links with Known Heuristic:

1. Construct a SharedValueModel of the board, measure the degree centrality of each node, create a queue where nodes are ordered according to descending degree centrality such that the nodes with the lowest centrality are dequeued first
2. Run Dancing Links as normal with one exception: rather than randomly selecting a node to guess values for, dequeue a node and assign it one of its possible values randomly

Experimental Procedure: For one of the above two algorithms, and one of the 13 Sudoku boards, run the algorithm to solve the board and record the runtime. 20 repetitions were made in total for each algorithm and board combination.

Hypothesis:

The hypothesis is that if the algorithm begins guessing values for nodes with low degree centrality in this model, it will minimize the number of future guesses to be made. In this way, the known heuristic for Dancing Links is implemented. It is expected that the mean runtime will be significantly reduced for each board when the heuristic is applied.

Results:

A two-sided t-test to test the hypothesis H_0 : the mean runtime is not changed once the heuristic is applied. A confidence level of 95% was used. It was found that for Sudoku boards 1-12, we were able to reject the null hypothesis very strongly. The results were unable to reject the null hypothesis for board 13. The p values and test statistics are displayed in Table I below:

Board	1	2	3	4	5	6	7	8	9	10	11	12	13
T-stat	-4.9	-4.9	-4.6	-6.9	-5.0	-5.3	-4.5	-5.7	-4.7	-3.5	-5.5	-4.5	-1.8
Pvalue	9e-5	9e-5	1e-4	7e-5	4e-5	2e-4	1e-5	1e-4	2e-3	2e-5	2e-4	3e-5	0.08

Table I: results of statistical testing for change in mean runtime after applying the heuristic

Shown below are boxplot diagrams for the runtimes. 20 repetitions are included in each boxplot.

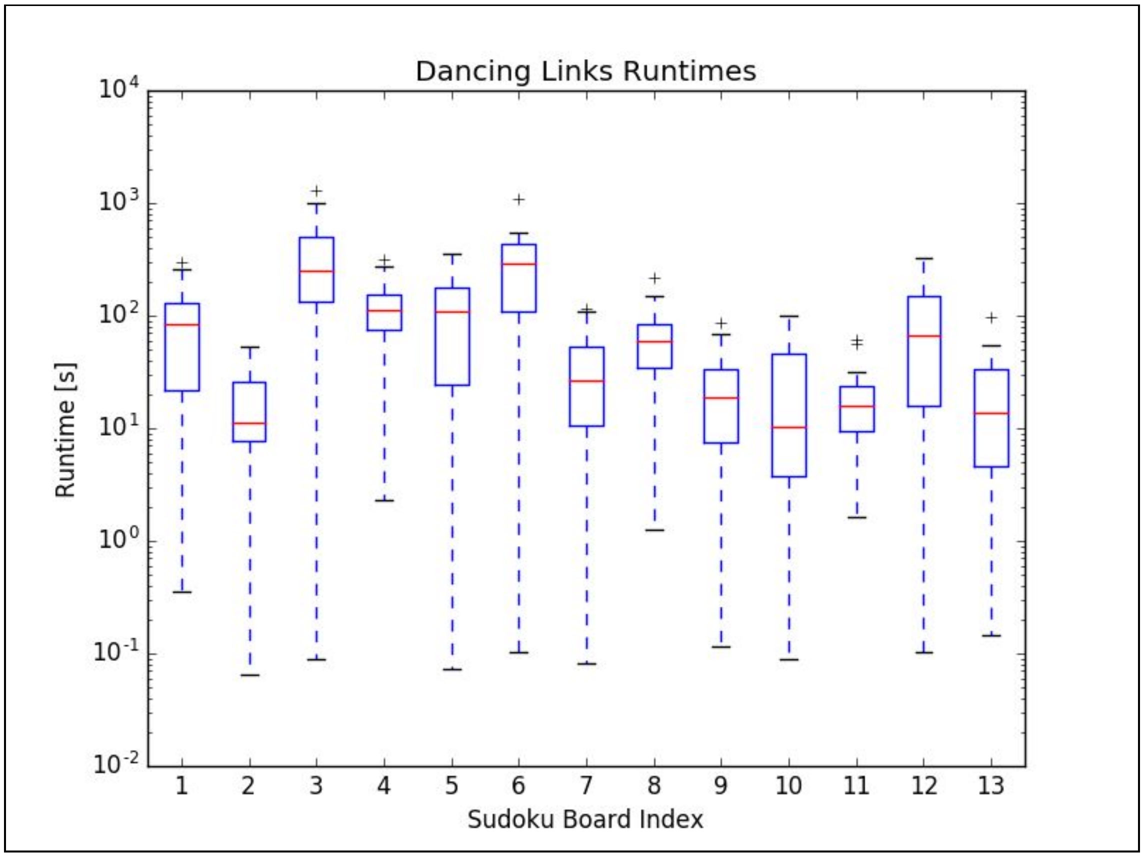


Figure 3: Dancing Links runtimes shown as boxplots for each of the 13 boards

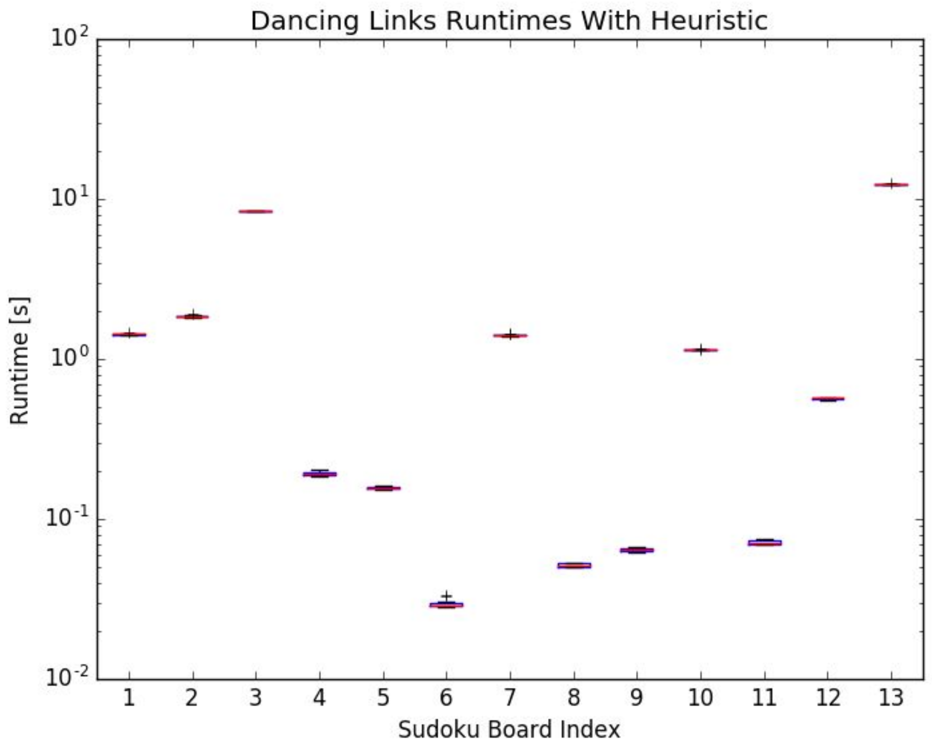


Figure 4: Dancing Links with applied heuristic runtimes shown as boxplots for each of the 13 boards

Discussion:

Based on the results shown above, network analysis has validated the only relevant heuristic to the Dancing Links algorithm. For the ordinary Dancing Links algorithm, runtimes can be large as as several minutes. This case occurs when the algorithm needs to exhaust large branches of the solution tree that are based on faulty guesses near the root. Because the ordinary Dancing Links algorithm randomly chooses the nodes it guesses a value for, it is much more likely that a poor choice will be made that contributes significantly to the runtime. The much higher chance of making a poor decision early in the Dancing Links algorithm also explains the much greater variance reflected in the boxplots of Figure 3. By comparison, the small variances in runtime for dancing links the applied heuristic can be explained by the fact the the queue produced by the SharedValueModel will always be the same for a given board. This means that the nodes are always chosen in the same order. With this in mind, interpretation of Figure 4 should be based on the position of the mean exclusively.

In verifying the heuristics to the Dancing Links algorithm, it has only demonstrated one intelligent order for processing the nodes based on the SharedValueModel. Future directions for our project would include formulating new meaning for graph metrics such as betweenness centrality and presence of communities, and formulating new intelligent orders for processing nodes. Additionally, new models similar to the SharedValueModel could be developed to elicit difference relationships between nodes and solution values using graph metrics. For example, a model could be constructed where nodes represent possible solution values for a each cell, and edges between two solution values represent compatibility of the two solutions. In order to make the graph more physically meaningful, edges would only be placed if two solution nodes belong to cells that share either a row, column, or sub-section of the Sudoku board. In such a graph, betweenness centrality could be considered a metric of independence of individual solutions rather than nodes (as in the case of the SharedValueModel). This would allow for the construction of a more nuanced Dancing Links algorithm that always guesses the most optimal solution for corresponding nodes at each step of the algorithm. It can be hypothesized that this algorithm would outperform our implementation of the heuristic.

Conclusion:

This project is successful in significantly improving the performance of the Dancing Links algorithm for solving Sudoku boards. This has been achieved by formulating several graphical representations of the Sudoku board to build a solver on top of as well as to measure which cells on the board were the best candidates for early guesses by the Dancing Links algorithm. The selected 13 boards were of the hardest possible class for 9x9 Sudoku boards, and the improvements in runtime were significant in twelve of the thirteen boards. When runtime was improved, it was often improved by a large amount. Some of the boards were solved in a fraction of a second, matching the best reported runtimes for boards of this difficulty. In order to achieve these results, a sudoku solver was developed from the ground up, capable of interfacing with igraph for optimizing the structure of the solution tree with respect to the network properties of our graphical representation. The framework developed is general and could be used for future research into optimizing the solution tree for graphical representations of non deterministic problems.

Works Cited:

[1] Knuth, D (1975). “Estimating the efficiency of backtrack programs,” Mathematics of Computation 29, 121–136.

[2] Knuth, D (2000). "Dancing links". Millennial Perspectives in Computer Science. P159. 187. arXiv:cs/0011047

[3] Floyd, R (1967) “Nondeterministic algorithms,” Journal of the ACM 14, 636–644.

[4] Golomb, S. Baumart, L (1965). “Backtrack programming,” Journal of the ACM 12, 516–524.

[5] McGuire, Gary (2012). "There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem." Web.

[6] Scott, D (1958) “Programming a combinatorial puzzle,” Technical Report No. 1 (Princeton, New Jersey: Princeton University Department of Electrical Engineering), ii + 14 + 5 pages.

[7] Kjell Ericson. Generate and Solve Sudoku <https://kjell.haxx.se/sudoku/>