Dancing Links Algorithm – Sudoku Solver

Colton Smith
Department of Computer Science
University of Kentucky
CS 315 - Algorithms
lilcoltsmith@gmail.com

ABSTRACT
The Dancing Links algorithm is an efficient implementation of the ideas explained in Knuth's Algorithm X. This algorithm is applied to problems known as *Exact Cover* problems, in which we try to find a way to cover a board to meet specifications. The Dancing Links Algorithm is a recursive, nondeterministic, depth-first, backtracking algorithm. In other words, it is used to solve those pesky NP-Complete problems.

CCS Concepts
• Knuth's Algorithm X, Dancing Links Algorithm, Exact Cover, NP-Complete

1. INTRODUCTION
Solving a Sudoku seems like a relatively easy concept, does it not? One could simply go through each square and try every possible combination. The problem here is that would be totally inefficient, given that there are 6,670,903,752,021,072,936,960 (6.67×10^{21}) total combinations for a standard Sudoku grid. While this is significantly less when you are given numbers, it is still way too many combinations to search through. Using the Dancing Links Algorithm, along side a smart auto complete algorithm, it is possible to solve puzzles in a matter of seconds.

2. Importance
Knuth says that most, if not all NP problems can be reduced to some form of Exact Cover problem. If this is the case, then using Dancing Links, we are able to come up with a solution to these problems efficiently. Whether or not they are an optimized solution is another story, but at least a solution can be reached. The three most common examples of problems related to this topic are:

- N Queens Problem
- Pentomino Tiling
- Sudoku

Other possible uses of Dancing Links would be for encryption and decryption keys. While these problems are all similar in terms of finding a solution, the approach for all of them differs greatly. The algorithm outlined by Knuth is just a guideline for an efficient way to process a grid/matrix to be parsed.

Conference'10, Month 1–2, 2010, City, State, Country.

DOI: http://dx.doi.org/10.1145/12345.67890

3. Sudoku Puzzles
There is a wide selection of Sudoku Puzzles. The standard grid has a size of 9 x 9, and they range up to 5 x 16 x 16 grids. For this project, only the standard 9 x 9 size grid was used for the Dancing Links Algorithm.

3.1 Sudoku Puzzle Rules
When solving a Sudoku Puzzle, there are of course rules to the game. When solving, each cell will contain a number between 1 and 9. The 9 x 9 grid is broken up into a smaller 3 x 3 grid as well. In each of these 3 x 3 grids, there may only be one of each numbers between 1 and 9. For each row, there may only be one of each number between 1 and 9. For each column, there may only be one of each number between 1 and 9. Only under these conditions can a Sudoku Puzzle be considered complete.

3.2 Difficulty of Sudoku Puzzles
There are three general difficulty levels for Sudoku Puzzles: Easy, Medium, and Hard. To solve these puzzles, there are three techniques that we can actually implement: deduction, making note of available values for each cell, and assumptions. These will be explained in detail during the Algorithm/Solution section.

4. Algorithm/Solution
As stated, Knuth's Dancing Links Algorithm is used to solve Exact Cover problems [1]. It is most commonly implemented using nodes, but the best way to describe this algorithm is when described as a tree. It is closely related to a simple depth-first search algorithm, in that it iterates through all possibilities until a solution is found. What makes Dancing Links special is the use of backtracking. When solving Sudoku Puzzles, we must create a 3-dimensional array in order to solve it more efficiently. We have the rows, columns, and for each cell (a combination of row and column), we have all possible numbers for that cell. Before we iterate through each cell and it's possible values, it is much more efficient to rule out all cells that have only one possibility based upon the provided grid (deduction method). The deduction method will be used as a heuristic. This alone is actually enough to solve most easy and medium difficulty Sudoku Puzzles. From the start, all cells are filled with possible values for that cell (1 – 9 for all cells initially), and it is updated as each cell if filled in. As a cell is filled in, simply clear the row, column, and sub-grid of the number selected. From here, we are left only to make assumptions. Now, iterate through the possible values of each cell, checking whether or not it is valid as you go. If it is a valid value, move to the next cell [2]. When all possible values have been tried, and none are valid, mark the current cell as 0 (the initial value), and backtrack. Remove the value already stored in the cell from possible answers and progress. These steps can be followed

recursively until we either reach the last cell or the first cell again. This means that the Sudoku puzzle has been solved.

5. Implementation

Python was the chosen language for the following implementation of the Dancing Links Algorithm. Following the algorithm outlined above, the pygame site-package for graphics, and the numpy site-package for building matrices, the following GUI was written.

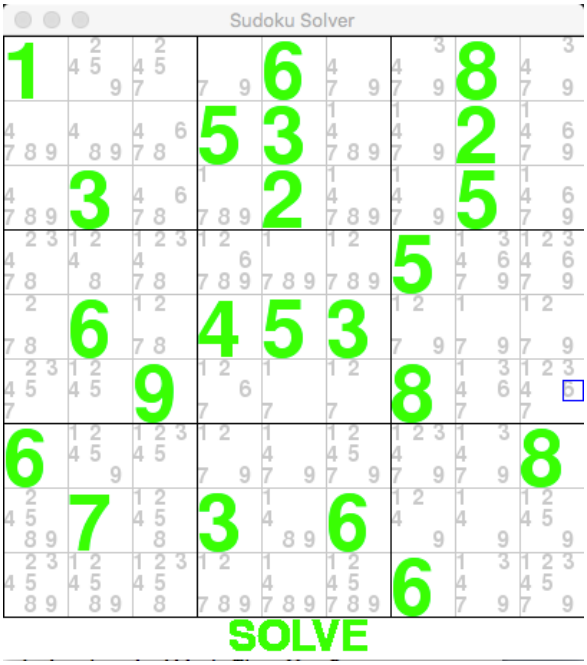

Figure 1. Sudoku Solver GUI

The user can input the Sudoku Puzzle they want to solve by clicking the number they wish to store into a cell. Upon doing this, the program will auto-complete any obvious choices using the deduction method previously explained, acting as a heuristic for the Dancing Links Algorithm. When the user has input the Sudoku Puzzle, and the puzzle has not been solved, the user can click solve at the bottom to run the Dancing Links Algorithm on the current Sudoku Board. The output will be sent to the terminal, and will consist of a completed Sudoku Grid followed by the run time of the Dancing Links Algorithm.

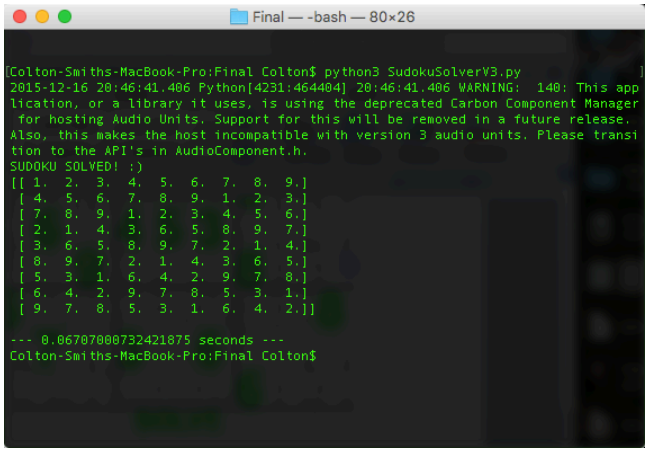

Figure 2. Solution output by Sudoku Solver

5.2 Run-Time for Different Difficulties

An important factor when looking at an algorithm's efficiency is the run time. In terms of O() notation, the run time of the Dancing Links Algorithm is: $O(n^m)$, where n is the number of possibilities per cell in the grid, and m being the number of empty squares. However, since the algorithm implemented has a heuristic, in which the cells contain information based on the row, column, and sub-grid, this number is greatly reduced.

Table 1. Run-Time for Different Puzzle Difficulties

	Easy	Medium	Hard
	.003407 sec	.070150 sec	1.20966 sec
	.004909 sec	.011032 sec	.017424 sec
	.004063 sec	.013887 sec	.887924 sec
Sum	μ = .004126 sec	μ = .031699 sec	μ = .705003 sec

6. REFERENCES

[1] Chu, J., A Sudoku Solver in Java implementating Knuth's Dancing Links Algorithm. *Harker Research Symposium* (April. 2006), DOI = https://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/sudoku.paper.html .

[2] Harrysson, M. and Laestander, H. 2014. *Solving Sudoku efficiently with Dancing Links.* Technical Report. KTH Computer Science and Communications. DOI = https://www.kth.se/social/files/54bda0d3f276541354ec0425/HLaestanderMHarrysson_dkand14.pdf .