Uniform Cost Sudoku Solver

Contents

[Purpose 2](#_Toc309916869)

[Rules for Sudoku 2](#_Toc309916870)

[Data Structures Used 2](#_Toc309916871)

[Program Files 2](#_Toc309916872)

[How the Program Works 3](#_Toc309916873)

[Results 3](#_Toc309916874)

[Conclusion 4](#_Toc309916875)

[Code Appendix](#_Toc309916875)

Michael Lowe

ECE478

Fall 2011

# Purpose

The purpose of this project was to use different search methods to solve a problem. I choose to use Uniform Cost search to solve Sudoku puzzles. I compared Uniform Cost with Depth First Search.

# Rules for Sudoku

Sudoku is a 9x9 grid filled with numbers. Each row, column and box must contain one of every number 1 to 9. The solver is given a puzzle where the grid is partially filled, and the solver is to fill in the rest. Typically with puzzles there is only one unique solution.

# Data Structures Used

There are multiple data structures used to solve this problem. Each Square in the Sudoku matrix is a struct that contains a value (representing the number in the square), a cost associated with filling in the square, and the squares coordinates in the matrix. The Sudoku matrix is a 3x3 of 3x3s of Sudoku squares. This makes it easy to traverse the Row, Column, and Boxes in the matrix. To make search possible I used a tree data structure. Each node in the tree is a Sudoku matrix with pointers to 9 possible matrixes, for each of the possible values to fill in a square. It also has a parent pointer that points to the previous node, this is maintained for backtracking.

# Program Files

I wrote this program in C++ using Visual Studios 2010. I started by creating a GUI that users can enter a puzzle into. The user puts given numbers into corresponding cells for the puzzle then hits “Solve”. Upon solving the puzzle I also have two fields for time it takes to solve it in milliseconds and number of “branches” the program takes to solve the puzzle. The program is divided into different .cpp files. There are a few to handle the GUI, these also perform tasks to fill in the initial matrix and get the time it takes solve the puzzle. The Sudoku.h and Sudoku.cpp files are used to define the data structures for the matrix as well as perform functions like finding the minimum cost square as well as performing an insert into the square. The sTree.h and sTree.cpp files are used for the tree data structure in the program.

Figure : Sudoku GUI

# How the Program Works

1. Cost is calculated for every square open square, the cost is the number of possible numbers to fill in the square. A filled square has a cost of 0.
2. If there is a 1 cost, it is automatically filled with the needed value. A parent pointer is set to the previous node.
3. If the cost is >1 one of the numbers is tried. A parent pointer is set to the node before the new node.
4. If there are no numbers to try move up one node and try another number at the previous node.

# Results

In addition to the Uniform Cost program I wrote, I modified the program slightly to not include cost in deciding which square to fill. This was the depth first search program I used to compare with my uniform cost search. I used puzzles from [www.websudoku.com](http://www.websudoku.com) ranked from easy to evil to compare the two programs. The results were as follows:

In most cases for time, besides easy, the Uniform Cost program outperformed the DFS program. The Uniform Cost function uses extra time in calculating the minimum cost for every node, but still in most cases it is the top performer. In the easy case it took more time to calculate these costs then to take the time to traverse these extra nodes. In all cases for the number of branches taken Uniform Cost is the top performer. In one of the cases it is interesting to note that the DFS program took 7200 branches to solve. This is one of the characteristics of depth first search, sometimes it can get lucky and take a short time to complete, other times it can get unlucky and take an extremely long time.

# Conclusion

In this comparison Uniform Cost is much more successful in solving Sudoku puzzles than Depth First Search. Even though it takes more time for calculations in this program the time it saves from traversing extra nodes makes it a more efficient program. Different search methods work better for different problems. Uniform Cost worked really well for this problem, but given more time I would like to have used more search methods to find if there are any better ways to solve Sudoku.