SudokuSolver

Github Repository

Solving sudoku using constraint propagation and search algorithms, taking input from the user in image files and processing it using a neural network model

Author: Dhruv Rai (@fLanKer42)

The solver program uses plain c++ to solve sudoku puzzles using non-learning CSP and Seacrh algorithms. The project includes a nodejs server that hosts a site through which users can post images of sudoku for it to solve. The image recognition involves open cv and a neural network model for recognising numbers.
The node server runs both python and cpp scripts as child processes in the server.

Introduction

Sudoku is a puzzle played on a partially filled 9x9 grid. The task is to complete the assignment using numbers from 1 to 9 such that the entries in each row, each column and each major 3x3 block are pairwise different. Like for many logical puzzles the challenge in Sudoku does not just lie in finding a solution. Well posed puzzles have a unique solution and the task is to find it without guessing, i.e. without search. For lesser posed problems or problems with higher difficulty, a search algorithm has been used
The basic Sudoku problem can be modelled with constraint programming by a combination of all different constraints.

Initial Solution

The initial solution uses a constraint model based on the rules of the game.
The row column constraint handles the interaction of rows and columns in the matrix. This reduces to a set of simple matching problems for a permutation matrix. Each value must occur exactly once in each row and column, this corresponds to a matching between row and columns (the two sets of nodes), and edges which link a row and a column if the given value is in the domain of the corresponding matrix element. By finding a maximal matching and then identifying strongly connected components in a re-oriented graph we can eliminate those values from all domains which do not belong to any maximal matching.
Similarly Row Block and Column Block interactions are used to eliminate values.

Constraint Satifaction Tree Problem:

Constraints

• Unary constraints or node constraints (eg. xi , 9)
• Binary constraints or edge between nodes (eg. xi , xj)
• Higher order or hyper-edge between nodes (eg. x1 + x2 = x3)

Node consistency

• For every variable Vi, remove all elements of Di that do not satisfy the unary constraints for the variable
• First step is to reduce the domains using node consistency

Arc consistency

• For every element xij of Di, for every edge from Vi to Vj, remove xij if it has no consistent value(s) in other domains satisfying the Constraints
• Continue to iterate using arc consistency till no further reduction happens.

Path consistency

• For every element yij of Di, choose a path of length L with L variables, use a consistency checking method similar to above to reduce domains if possible

---

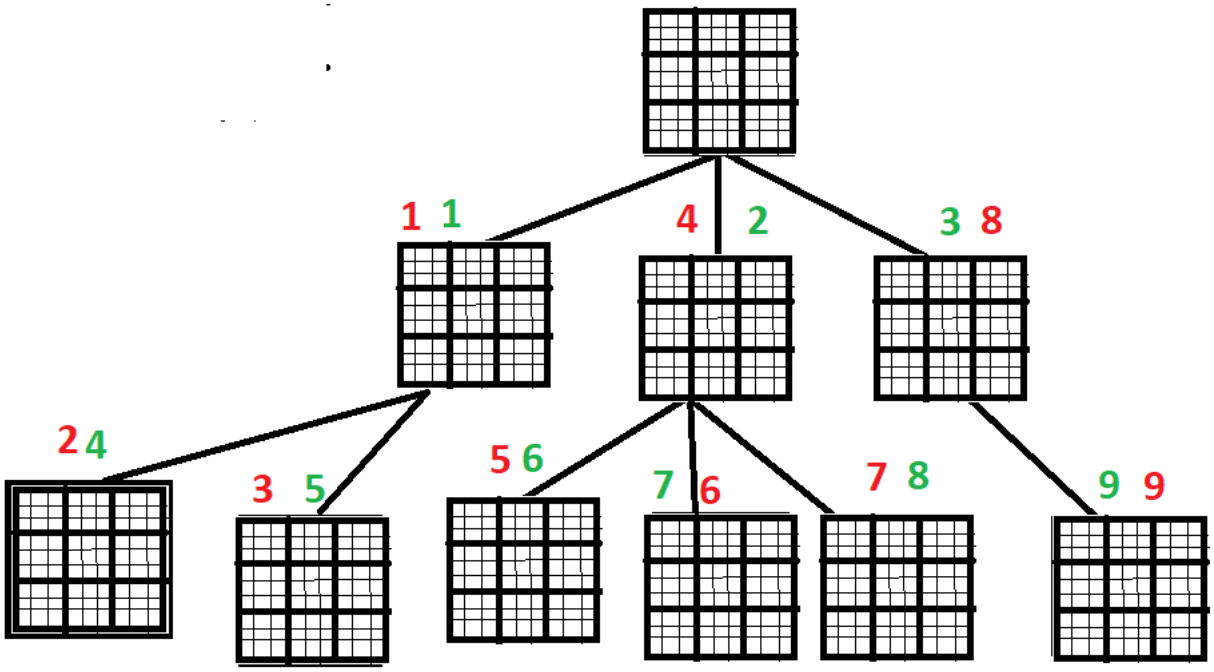
But what if the CSP method fails to complete the puzzle?
In such cases, we have to stick to AI search methods like BFS, DFS, IDS, DFBB, A* and IDA.

Problem Formulation by AI Search Methods consists of the following key concepts

– Configuration or State
– Constraints or Definitions of Valid Configurations
– Rules for Change of State and their Outcomes
– Initial or Start Configurations
– Goal Satisfying Configurations
– An Implicit State or Configuration Space
– Valid Solutions from Start to Goal in the State Space
– General Algorithms which SEARCH for Solutions in this State Space

Example of a single node in the state space.

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



Plain BFS in green and DFS in red. Methods for searching through the tree.