Non-Deterministic Sudoku Solver

By Binod Timalsina & Khadeeja Din

			4
	2	1	
	1	4	
			1

A 4x4 Sudoku Puzzle

1	3	2	**4**
4	**2**	**1**	3
3	**1**	**4**	2
2	4	3	**1**

Solution achieved

Sudoku Description

Sudoku is a logic based, combinatorial number-placement puzzle of size nxn.

The puzzle is composed of a nxn grid containing nxn cells.

The objective of the game is to organize the numbers 1-n in such a way that all the n rows, n columns, and n sub-grids have 1-n distinct numbers each.

Design Problem Description

Sudoku is an NP-complete problem

logical algorithms that do solve sudoku, are only guaranteed to solve the puzzles specifically constructed for them following the logical rules alone.

We have picked the non deterministic method for addressing the sudoku problem

to explore how efficient or inefficient a non-deterministic Sudoku solver can be.

Reference from Text

In nondeterministic Sudoku, evaluation of choices may result in the discovery of a dead end, in which case evaluation must backtrack to a previous choice point.

Use of Amb Form to support this idea.

Design Outline

1. Fill all the empty spaces of sudoku by assigning value such that each sub grid has 1 to n numbers. Consider these cells movable cells.
2. randomly select a subgrid and randomly select two movable grids from that subgrid and swap their values.
3. Repeat swap until the problem is solved

Problem Approach

Our work for this problem consisted of two main parts:

1. designing the data structure for puzzle and writing its various functions.
2. designing a solution that keeps track of error and makes use of amb.

Design Process

1. Fill Sudoku

One of the most complicated part of our function was filling the empty spots such that each sub grid have all the numbers from 1 to n for a n × n Sudoku. S

(0, 0)	(0,1)	(0, 2)	(0, 3)
(1, 0)	(1,1)	(1, 2)	(1, 3)
(2, 0)	(2,1)	(2, 2)	(2, 3)
(3, 0)	(3,1)	(3, 2)	(3, 3)

(0, 0)	(0,1)
(1, 0)	(1,1)
(2, 0)	(2,1)
(3, 0)	(3,1)
(0, 2)	(0, 3)
(1, 2)	(1, 3)
(2, 2)	(2, 3)
(3, 2)	(3, 3)

we split the list into n places to get list sorted by sub grid.

(0, 0)	(0,1)	(1, 0)	(1,1)
(2, 0)	(2,1)	(3, 0)	(3,1)
(0, 2)	(0, 3)	(1, 2)	(1, 3)
(2, 2)	(2, 3)	(3, 2)	(3, 3)

Design Process

Using the sort function we fill the Sudoku. First we replace all empty spaces with their coordinates.

x	x	x	4
x	2	1	x
x	1	4	x
x	x	x	1

(0, 0)	(0,1)	(1, 0)	4
(2, 0)	2	1	(3,1)
(0, 2)	1	4	(1, 3)
(2, 2)	(2, 3)	(3, 2)	1

Design Process

Swapping Numbers:

The solution uses a random-swap function which will randomly select two coordinates from a subgrid to be swapped.

The swap operation will then return the puzzle with the values for those coordinates switched.

The swapping is done one coordinate at a time.

The row of the coordinate is extracted, and then the value at the coordinates column position in the row is changed.

The row is then appended back into its position, and the puzzle is set to the new puzzle with changed value.

Design Process

Calculating error:

The error in Sudoku is number of repetitions in row, columns and sub grid. Since we fill the Sudoku so that there is no repetition in sub grid we only need to account for errors in rows and columns.

To calculate error in rows we simply count the number of repetitions in a row. We map this function on Sudoku so that we get a list of errors for each row. Similarly we calculate column errors by first transposing the Sudoku and calculating row error for transposed Sudoku.

Design Process

Solving the Puzzle:

The algorithm for solving this Sudoku problem is (Lewis):

Fill all the empty spaces of Sudoku by assigning value such that each sub grid has 1 to n numbers. Consider these cells movable cells.

randomly select a sub grid and randomly select two movable grids from that sub grid and swap their values.

Repeat swap until the problem is solved

Technology Used

Amb

Give a list of options: Then (an-element of (list a) gives 1st element of list.

- Assert (condition) if not return amb.
- i.e. discard procedure up top now and restart using next item from list a
- Chronological backtracking

Technology contd.

Infinite Streams

Example:

a pair: (val.<stream>) so we select 1st use it and if we need more number execute <stream> function that gives next pair of (val.<stream>)

So can run calculation infinitely.

Sample Output Solution

```
(define sodoku (list (list 2 4 3 1)
                     (list x x x x)
                     (list x 1 2 x)
                     (list 3 x x x)))
```

```
(2 4 3 1)-0
(3 1 4 2)-0
(4 1 2 4)-1
(3 2 3 1)-1
------------------------------
(1 1 1 1)Total error: 6


(2 4 3 1)-0
(1 3 4 2)-0
(4 1 2 4)-1
(3 2 3 1)-1
------------------------------
(0 0 1 1)Total error: 4


(2 4 3 1)-0
(1 3 4 2)-0
(4 1 2 4)-1
(3 2 1 3)-1
------------------------------
(0 0 0 0)Total error: 2

((2 4 3 1) (1 3 4 2) (4 1 2 3) (3 2 1 4))
>
```

What Went Wrong

Dead end

1	4	3	**4**	1
3	**2**	**1**	2	1
2	**1**	**4**	3	0
4	3	2	**1**	0
0	0	0	0	2

1	/3	/2	**4**	/0
/4	**2**	**1**	/3	/0
2	**1**	**4**	3	0
4	3	2	**1**	0
1	1	1	1	4

So there is no swap that decreases total error. Every swap increases error in column while decreasing in row.

There is no way a swap would decrease error at this stage. We need to solve this logically by allowing error to increase.

Error contd.

Some time we need to back track and re-swap some stage differently to get a error

1	/3	/2	**4**	/0
/4	**2**	**1**	/3	/0
2	**1**	**4**	/2	1
/3	/4	/3	**1**	1
0	0	0	0	2

1	/3	/2	**4**	/0
/4	**2**	**1**	/3	/0
/3	**1**	**4**	/2	0
/2	/4	/3	**1**	0
0	0	0	0	0

Future Development

For Sudoku we need to Create a dead end checker that checks if we have hit a dead end and allows us to swap even if there is error. We could make it so that if n in (n X n Sudoku) swaps does not decrease error restart the algorithm.