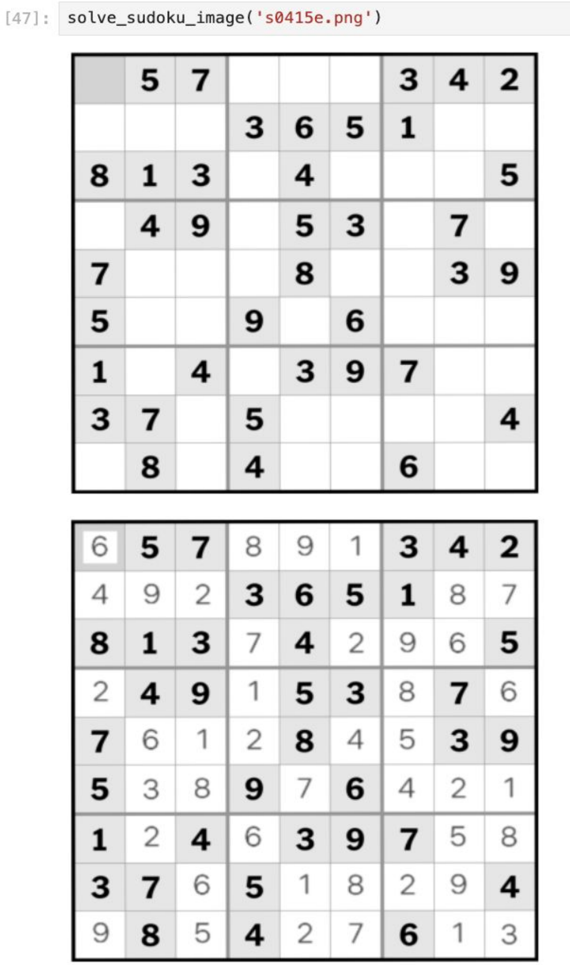
explorations in AI...

Computer Vision Sudoku Solver

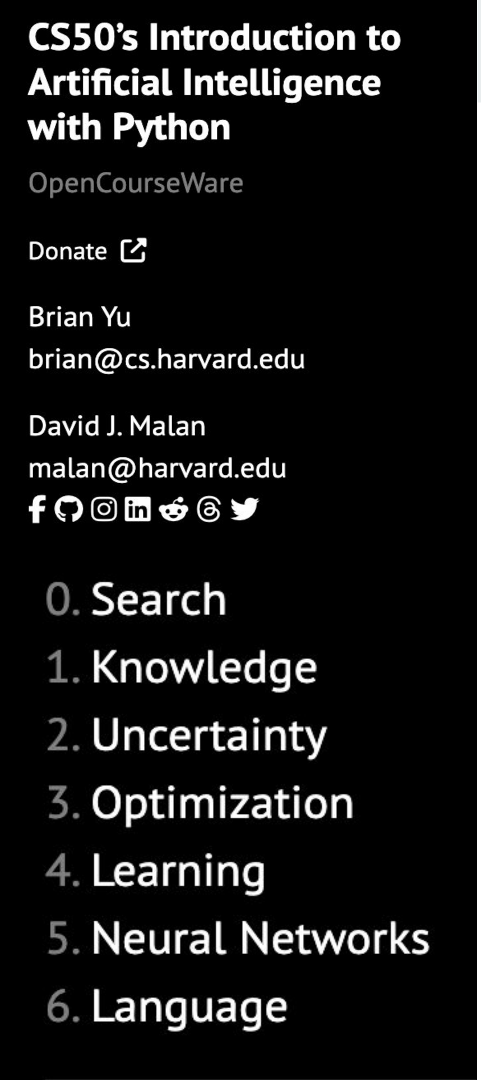
Edward Lee
eclee@ccsf.edu



```
[47]: solve_sudoku_image('s0415e.png')
```

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

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



CS50's Introduction to Artificial Intelligence with Python

OpenCourseWare

Donate

Brian Yu
brian@cs.harvard.edu

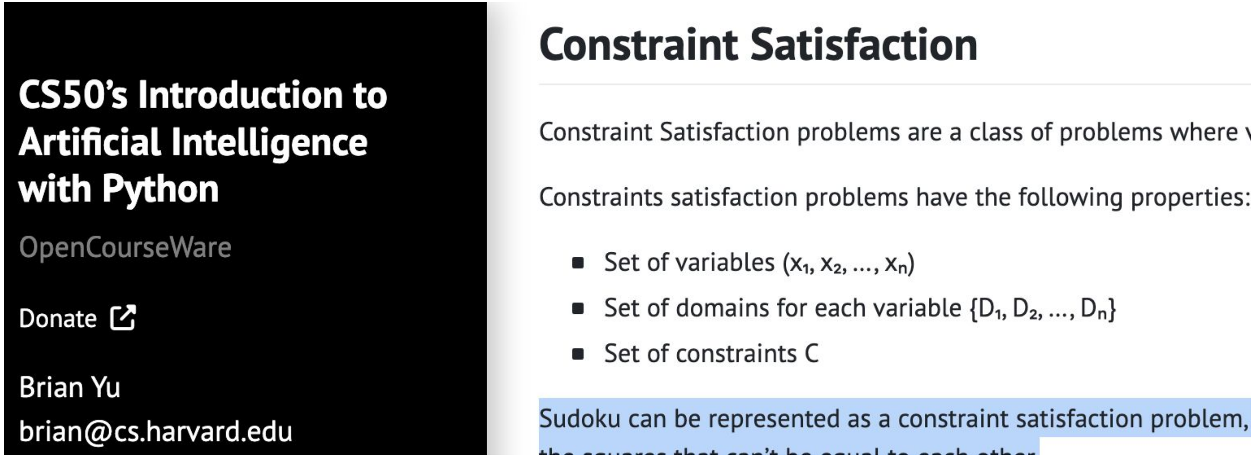
David J. Malan
malan@harvard.edu

0. Search
1. Knowledge
2. Uncertainty
3. Optimization
4. Learning
5. Neural Networks
6. Language

← Great (free) Online Class on AI

cs50.harvard.edu/ai/2024/notes

- Foundation provided by
- Math 108, 110, 115, 120
  - regression, derivatives, minimum of a function
  - state machine, graphs, recursion, logic
  - vector, dot product, matrix multiplication
- CS 231, 111C
  - python, jupyter notebooks
  - data structures
  - trees, graphs, stack, depth first search



CS50's Introduction to Artificial Intelligence with Python

OpenCourseWare

Donate

Brian Yu
brian@cs.harvard.edu

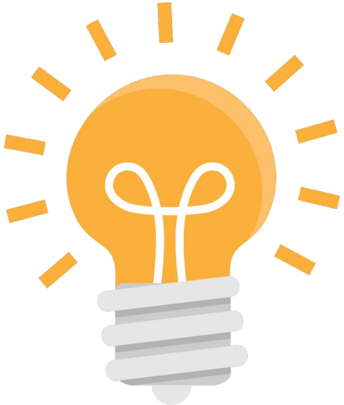
Constraint Satisfaction

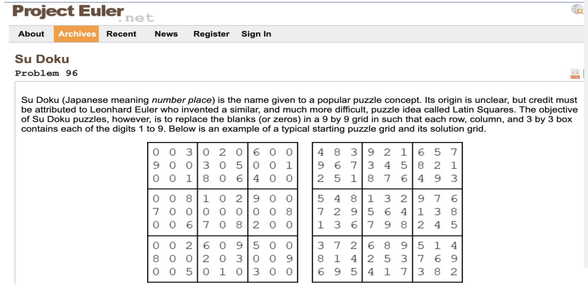
Constraint Satisfaction problems are a class of problems where

Constraints satisfaction problems have the following properties:

- Set of variables ($x_1, x_2, ..., x_n$)
- Set of domains for each variable $\{D_1, D_2, ..., D_n\}$
- Set of constraints C

Sudoku can be represented as a constraint satisfaction problem,
the squares that can't be equal to each other





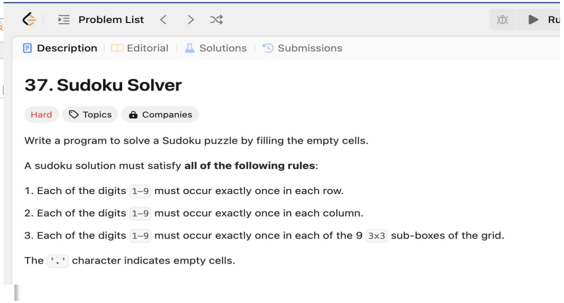
Su Doku

Problem 96

Su Doku (Japanese meaning *number place*) is the name given to a popular puzzle concept. Its origin is unclear, but credit must be attributed to Leonhard Euler who invented a similar, and much more difficult, puzzle idea called Latin Squares. The objective of Su Doku puzzles, however, is to replace the blanks (or zeros) in a 9 by 9 grid in such that each row, column, and 3 by 3 box contains each of the digits 1 to 9. Below is an example of a typical starting puzzle grid and its solution grid.

0	0	3	0	2	0	6	0	0
9	0	0	3	0	5	0	0	1
0	0	1	8	0	6	4	0	0
0	0	8	1	0	2	9	0	0
7	0	0	0	0	0	0	0	8
0	0	6	7	0	8	2	0	0
0	0	2	6	0	9	5	0	0
8	0	0	2	0	3	0	0	9
0	0	5	0	1	0	3	0	0

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



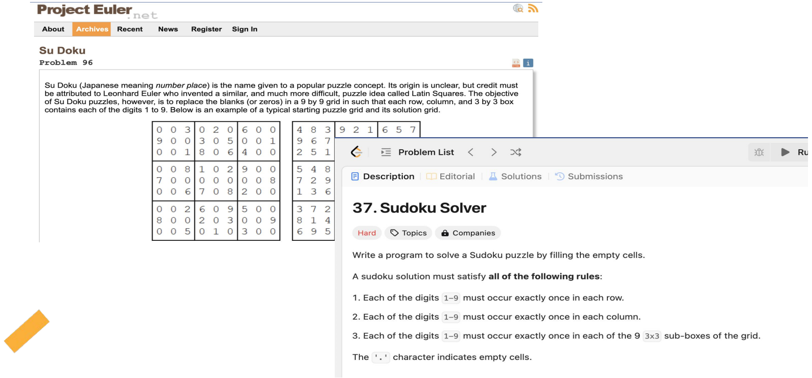
37. Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

The `'.'` character indicates empty cells.



Su Doku

Problem 96

Su Doku (Japanese meaning *number place*) is the name given to a popular puzzle concept. Its origin is unclear, but credit must be attributed to Leonhard Euler who invented a similar, and much more difficult, puzzle idea called Latin Squares. The objective of Su Doku puzzles, however, is to replace the blanks (or zeros) in a 9 by 9 grid in such that each row, column, and 3 by 3 box contains each of the digits 1 to 9. Below is an example of a typical starting puzzle grid and its solution grid.

0	0	3	0	2	0	6	0	0
9	0	0	3	0	5	0	0	1
0	0	1	8	0	6	4	0	0
0	0	8	1	0	2	9	0	0
7	0	0	0	0	0	0	0	8
0	0	6	7	0	8	2	0	0
0	0	2	6	0	9	5	0	0
8	0	0	2	0	3	0	0	9
0	0	5	0	1	0	3	0	0

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

37. Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

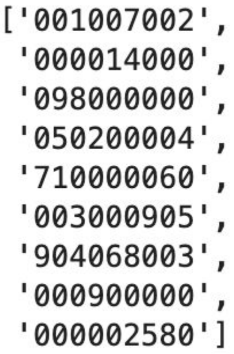
The `'.'` character indicates empty cells.

Discrete Mathematics Math115 - graphs, state machine, sets

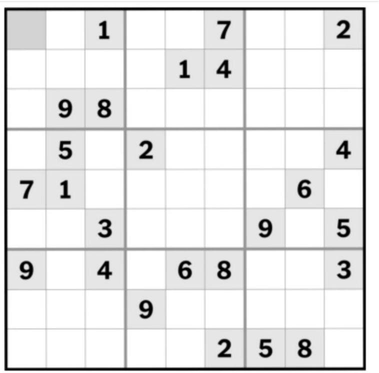
Data Structures CS111C - depth first search, recursion, stacks/queues

Became 19,000 th person to submit the correct answer.

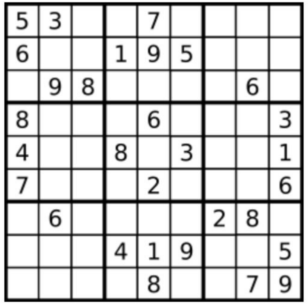
Sudoku
text based character entry vs image recognition



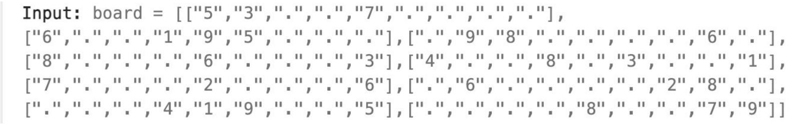
```
['001007002',
 '000014000',
 '098000000',
 '050200004',
 '710000060',
 '003000905',
 '904068003',
 '000900000',
 '000002580']
```



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



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



```
Input: board = [["5","3",".",".","7",".",".",".","."],
["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],
["8",".",".",".","6",".",".",".","3"],["4",".",".","8",".","3",".",".","1"],
["7",".",".",".","2",".",".",".","6"],[".","6",".",".",".",".","2","8","."],
[".",".",".","4","1","9",".",".","5"],[".",".",".",".","8",".",".","7","9"]]
```

Project Euler and Leetcode challenges use non-human text based input…

HOW DULL!!!

Lets input the board from a screenshot image

Lets code!

https:// bit.ly/sudo_vision

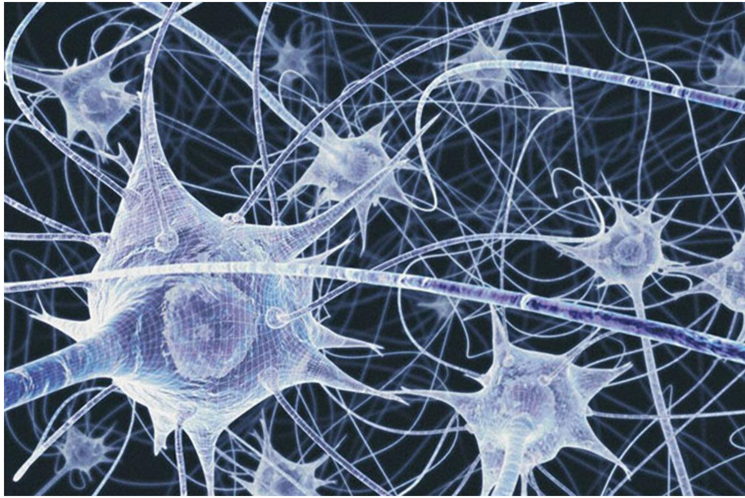
2 Flavors of AI discussed today:

1) neural networks - for computer vision to recognize a Sudoku board
2) Inference, knowledge, search - for puzzle solving logic

Neural Networks - works well for Image Recognition

Called neural network because the original was inspired by the biological structure of neurons.

Convolutional neural networks are a variant that is widely used in image recognition.



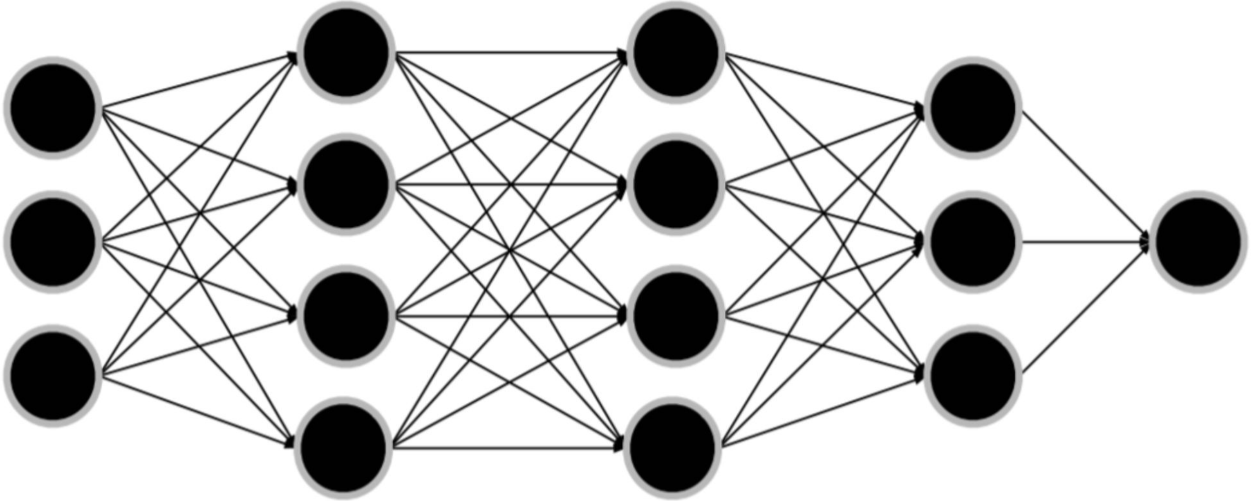
Think of neural networks as a form of linear regression on steroids:

Try to find a best fit ~~line~~ function to the data that you have and then use that ~~line~~ function to make predictions.

Regression on Steroids: ~~Line~~ (Function) of Best Fit

Basic Math behind Neural Networks…

x_1 v_2 v_3 v_4 v_5



input $v_2=f_2(Ax_1)$ $v_3=f_3(Bv_2)$ $v_4=f_4(Cv_3)$ output

Loss Function
Examples are:
Mean Squared Error
Mean Absolute Error

Matrix multiplication connects every layer of the graph
Every node is a dot product of the previous layer combined with an activation function
The activation function between layers makes model non linear and suited for image recognition
These matrices can be very large…. our input layer has 800 pixels,

Neural networks need training?

- Called “supervised learning”.
- We need a set of labeled training data to tune our neural network

Neural networks need to be trained?

- Called “supervised learning”.
- We need a set of labeled training data to tune our neural network
- **We start with a guess about the matrices that connect the layers.**
- **We pass data through the model and calculate a loss compared to what we know is true**

Neural networks need to be trained?

- Called "supervised learning".
- We need a set of labeled training data to tune our neural network
- We start with a guess about the matrices that connect the layers.
- We pass data through the model and calculate a loss compared to what we know is true
- **We use gradient descent (follow the derivative) to minimize the loss.**
- **The algorithms used today involve random processes, so our results will vary**
- **We repeat this process many times over our data set and hope for convergence**

Neural networks need to be trained?

- Called “supervised learning”.
- We need a set of labeled training data to tune our neural network
- We start with a guess about the matrices that connect the layers.
- We pass data through the model and calculate a loss compared to what we know is true
- We use gradient descent (follow the derivative) to minimize the loss.
- The algorithms used today involve random processes, so our results will vary
- We repeat this process many times over our data set and hope for convergence
- **Risk of “overfitting” to training data, which results in a loss of generality to new data**

Lets code!

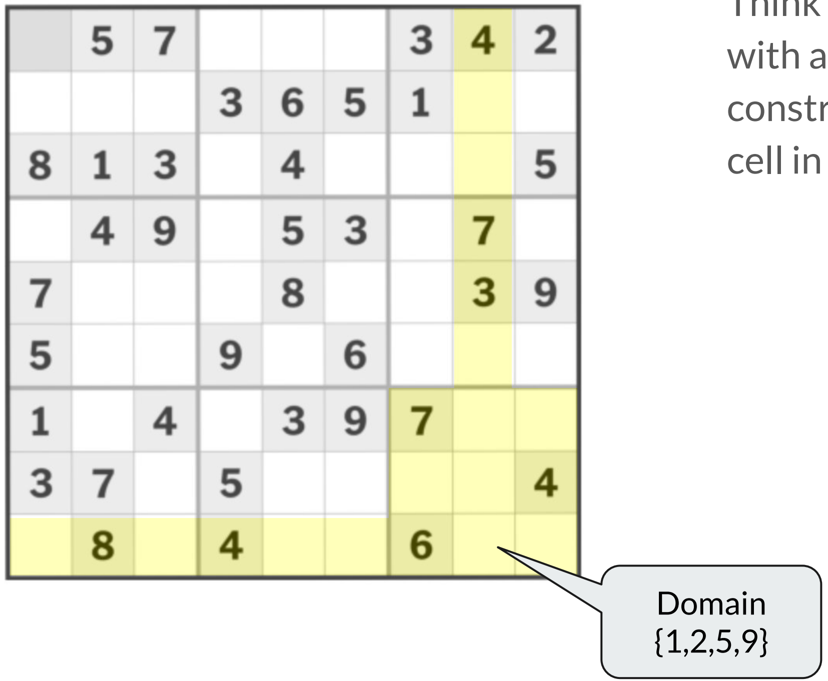
https:// bit.ly/sudo_vision

A different kind of AI to solve a puzzle

- Search
- Graphs, neighbors
- Constraint satisfaction
- Knowledge base
- Inference
- Recursion
- Depth first
- Backtracking

Now that we've seen it, how to solve it?

```
b=read_sudoku_board('s0415e.png')
```



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

Domain
{1,2,5,9}

Keywords: **Constraint satisfaction, Inference, Domain, Set Theory**

Think of each empty cell on the board as a variable with a domain of possible values (1-9). There is a constraint that a cell can't share the value with another cell in its row, column or block

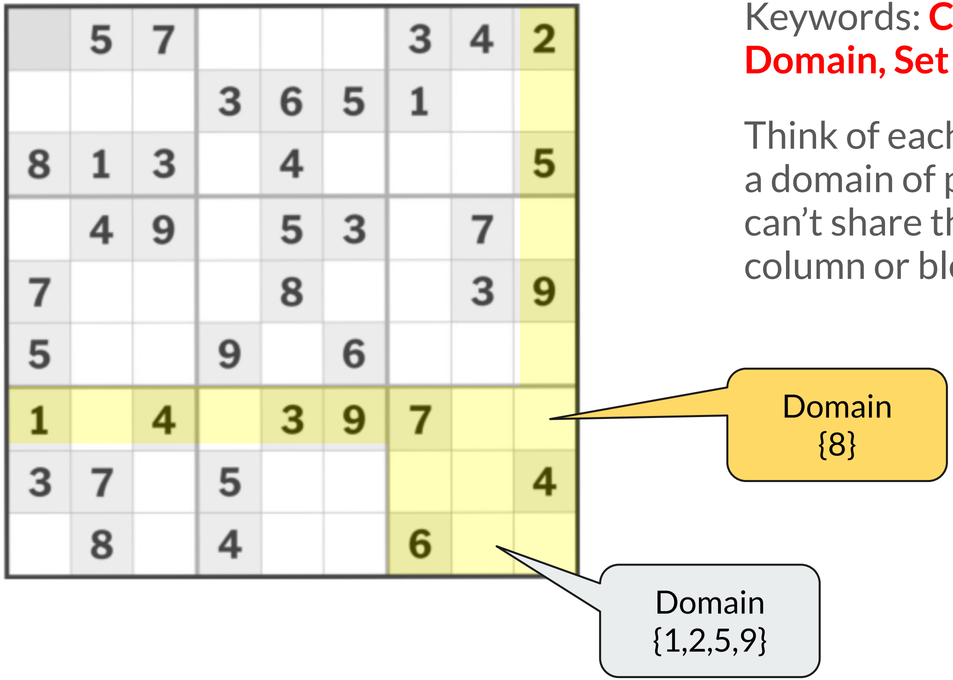
Set operations are useful in domain calculations!

$$\mathbf{D(8,7) = (set(row)\ U\ set(column)\ U\ set\ (block))^{C}}$$

$$= \{4,6,8\}\ U\ \{3,4,7\}\ U\ \{4,6,7\})^{C}$$
$$= \{3,4,6,7,8\}^{C}$$
$$= \{1,2,5,9\}$$

How to solve a sudoku once the computer recognizes the board... different kind of AI

```
b=read_sudoku_board('s0415e.png')
```



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

Domain
{8}

Domain
{1,2,5,9}

Keywords: **Constraint satisfaction, Inference, Domain, Set Theory**

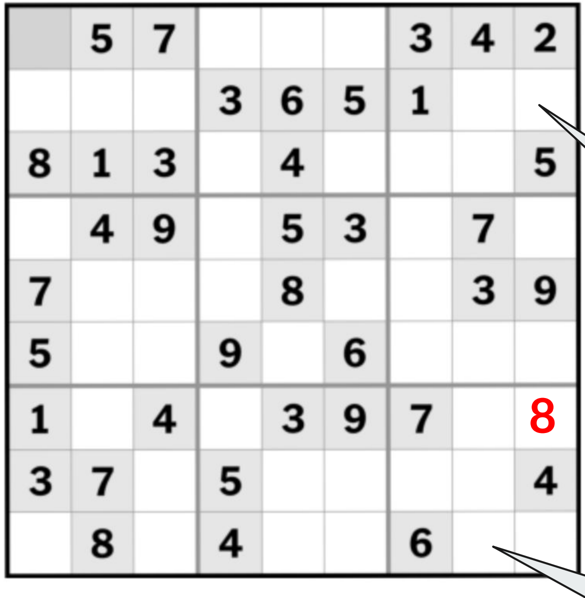
Think of each cell on the board as a variable with a domain of possible values (1-9). But each cell can't share the value with another cell in its row, column or block

81 cells, each with a domain

Inferences can be made when the domain of a cell contains a single value. This cell must be an 8 because it's the only possible value.

Inference changes the state of your knowledgebase...

```
b=read_sudoku_board('s0415e.png')
```

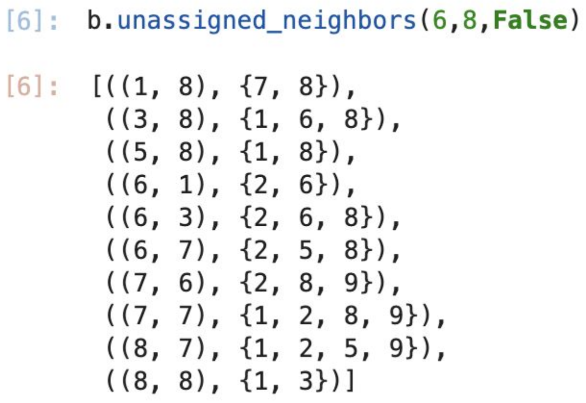


	5	7				3	4	2
			3	6	5	1		
8	1	3		4				5
	4	9		5	3		7	
7				8			3	9
5			9		6			
1		4		3	9	7		8
3	7		5					4
	8		4			6		

Domain
Before {7,8}
After {7}

Domain
Before {1,2,5,9}
After unchanged

Each inference made to a cell can affect the domains of its neighbors and may lead to new inferences.

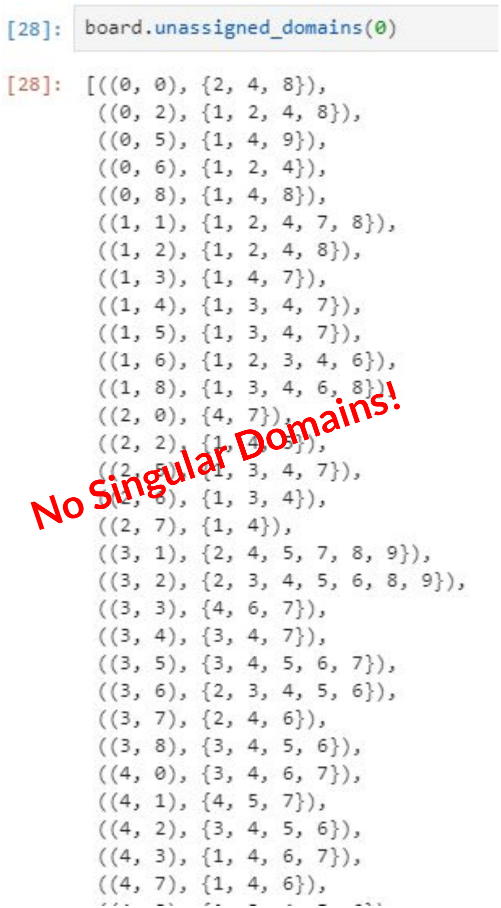


```
[6]:  b.unassigned_neighbors(6,8,False)

[6]:  [((1, 8), {7, 8}),
       ((3, 8), {1, 6, 8}),
       ((5, 8), {1, 8}),
       ((6, 1), {2, 6}),
       ((6, 3), {2, 6, 8}),
       ((6, 7), {2, 5, 8}),
       ((7, 6), {2, 8, 9}),
       ((7, 7), {1, 2, 8, 9}),
       ((8, 7), {1, 2, 5, 9}),
       ((8, 8), {1, 3})]
```

Lets code!

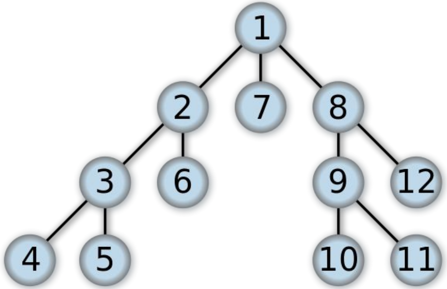
https:// **bit.ly/sudo_vision**

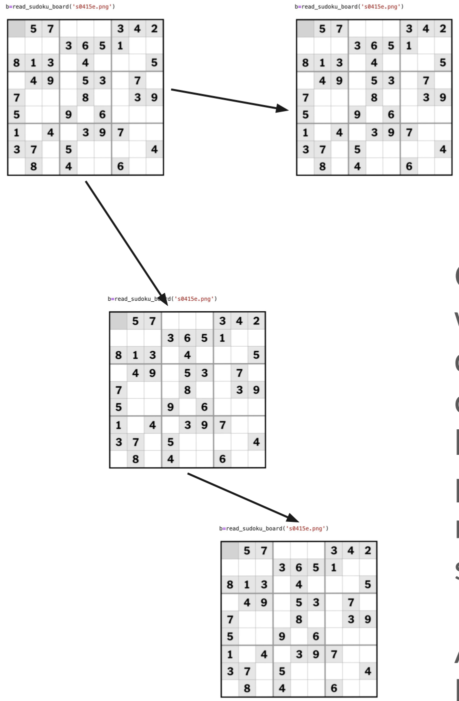
Inference may only go so far. Implement a search



```
[28]: board.unassigned_domains(0)

[28]: [((0, 0), {2, 4, 8}),
        ((0, 2), {1, 2, 4, 8}),
        ((0, 5), {1, 4, 9}),
        ((0, 6), {1, 2, 4}),
        ((0, 8), {1, 4, 8}),
        ((1, 1), {1, 2, 4, 7, 8}),
        ((1, 2), {1, 2, 4, 8}),
        ((1, 3), {1, 4, 7}),
        ((1, 4), {1, 3, 4, 7}),
        ((1, 5), {1, 3, 4, 7}),
        ((1, 6), {1, 2, 3, 4, 6}),
        ((1, 8), {1, 3, 4, 6, 8}),
        ((2, 0), {4, 7}),
        ((2, 2), {1, 4, 5}),
        ((2, 5), {1, 3, 4, 7}),
        ((2, 6), {1, 3, 4}),
        ((2, 7), {1, 4}),
        ((3, 1), {2, 4, 5, 7, 8, 9}),
        ((3, 2), {2, 3, 4, 5, 6, 8, 9}),
        ((3, 3), {4, 6, 7}),
        ((3, 4), {3, 4, 7}),
        ((3, 5), {3, 4, 5, 6, 7}),
        ((3, 6), {2, 3, 4, 5, 6}),
        ((3, 7), {2, 4, 6}),
        ((3, 8), {3, 4, 5, 6}),
        ((4, 0), {3, 4, 6, 7}),
        ((4, 1), {4, 5, 7}),
        ((4, 2), {3, 4, 5, 6}),
        ((4, 3), {1, 4, 6, 7}),
        ((4, 7), {1, 4, 6}),
```

No Singular Domains!





b=read_sudoku_board('s0415e.png')

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

b=read_sudoku_board('s0415e.png')

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

b=read_sudoku_board('s0415e.png')

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

b=read_sudoku_board('s0415e.png')

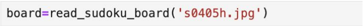
	5	7				3	4	2
			3	6	5	1		
8	1	3		4				5
	4	9		5	3		7	
7				8			3	9
5			9		6			
1		4		3	9	7		
3	7		5					4
	8		4			6		

Our graph will be very dense… could be hundreds of potential nodes to search.

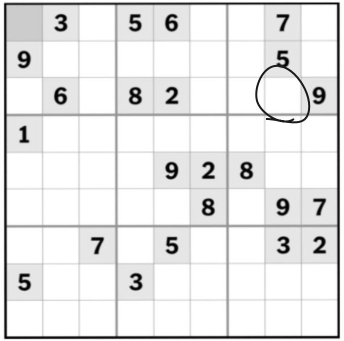
Also need to keep track of state when we backtrack

Sometimes you have to guess... Search!

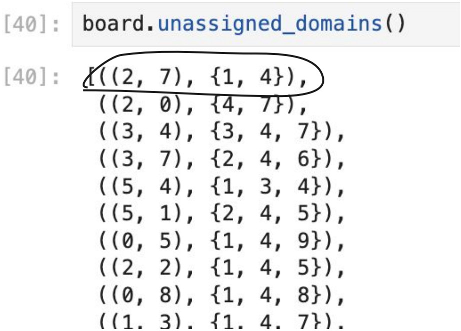
Search tree pruning: Let’s pick from the smallest domain!



```
board=read_sudoku_board('s0405h.jpg')
```



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



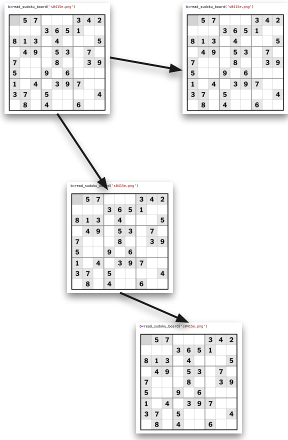
```
[40]: board.unassigned_domains()

[40]: [((2, 7), {1, 4}),
       ((2, 0), {4, 7}),
       ((3, 4), {3, 4, 7}),
       ((3, 7), {2, 4, 6}),
       ((5, 4), {1, 3, 4}),
       ((5, 1), {2, 4, 5}),
       ((0, 5), {1, 4, 9}),
       ((2, 2), {1, 4, 5}),
       ((0, 8), {1, 4, 8}),
       ((1, 3), {1, 4, 7}),
```

Domain for (2,7) is {1,4}... here we guess because we have a 50% chance and if we fail we only have one other option to try.

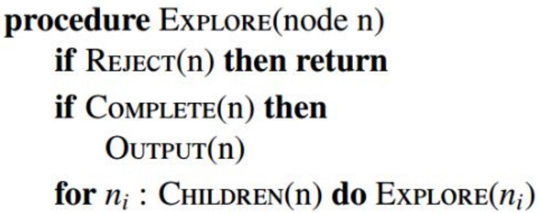
We assign cell value 1 and proceed from there. If we find this leads to a dead end, we backtrack and assign 4.

Math 115/ CS111C: State Machines, Graphs, DFS

- Imagine the Sudoku game as sequence of filling in empty cells.
- You transition from board (state) to board by filling a cell.
- It’s a graph! Vertices: board, Edges: fill a cell
- Graphs are useful because there are algorithms, one called backtracking DFS, we can use when constraints are involved.

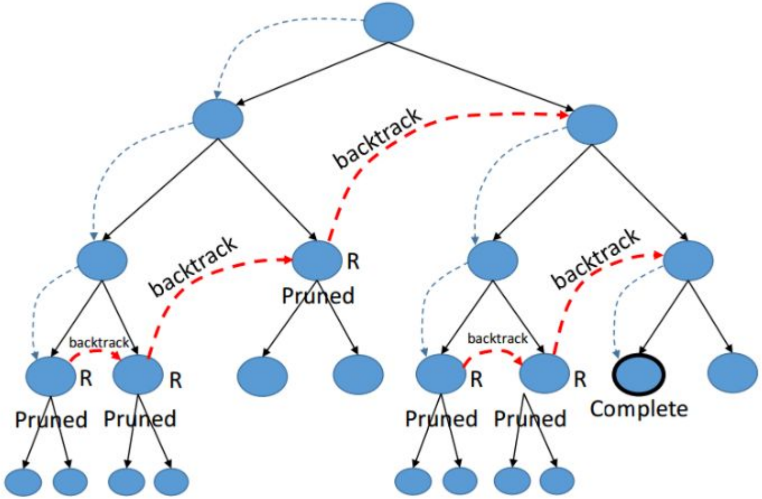


Backtracking Search - Form of DFS useful for constraint satisfaction problems



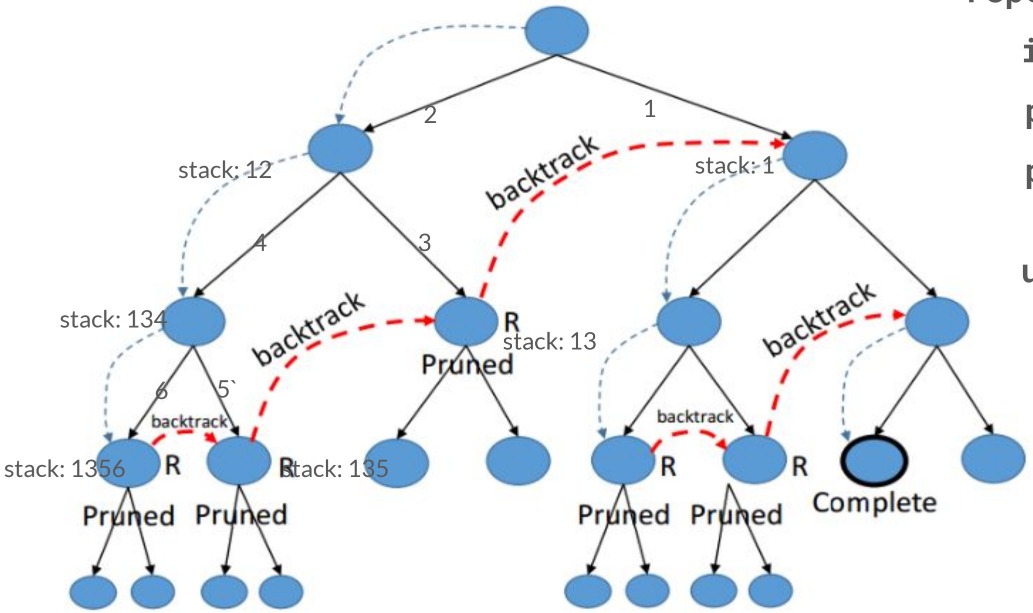
```
procedure EXPLORE(node n)
    if REJECT(n) then return
    if COMPLETE(n) then
        OUTPUT(n)
    for $n_i$ : CHILDREN(n) do EXPLORE($n_i$)
```

← A recursive DFS



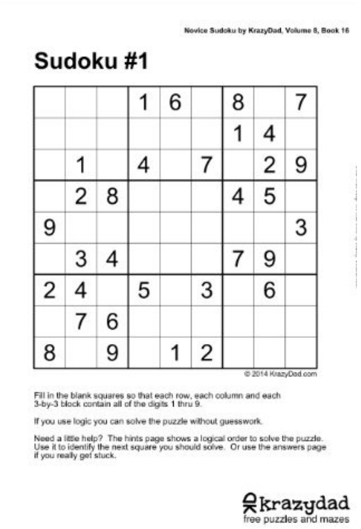
Backtracking Search - Form of DFS useful for constraint satisfaction problems

CS111C -
implement recursion with a stack



```
function dfs_search(board):
    create an empty stack
    repeat:
        if board is complete, return the board
        place edges from smallest domain on stack
        pull edges off stack until you find one
              that satisfies constraints
        use the edge to update the board
```

Input Image File



Novice Sudoku by KrazyDad, Volume 8, Book 16

Sudoku #1

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

© 2014 KrazyDad.com

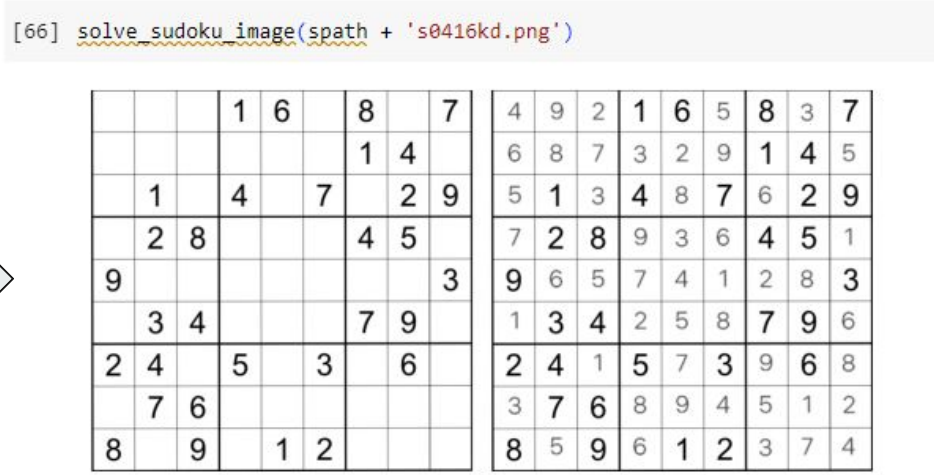
Fill in the blank squares so that each row, each column and each 3-by-3 block contain all of the digits 1 thru 9.

If you use logic you can solve the puzzle without guesswork.

Need a little help? The hints page shows a logical order to solve the puzzle. Use it to identify the next square you should solve. Or use the answers page if you really get stuck.

krazydad
free puzzles and mazes

Output Image



```
[66] solve_sudoku_image(spath + 's0416kd.png')
```

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

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

Questions?