Automatically Generating Puzzle Problems with Varying Complexity

Amy Chou and Justin Kaashoek
Mentor: Rishabh Singh

Fourth Annual PRIMES MIT Conference
May 19th, 2014

The Motivation

- We want to help people learn programming!
- To learn, people want many examples of different complexity

Current Situation

- Homework problems are few and fixed difficulty
- Online courses such as 6.00x do not have an efficient way to check interesting problems

```
new_hand = self.hand.copy()
for letter in word:
    try:
        new_hand[letter] -= 3
    except KeyError:
        # if 'letter' isn't in the hand, we can't make the word from this hand.
        return False
for letter in new_hand.keys():
    # If any of the letter counts of the new hand are less than zero after the
    # update, then we can't make the word from this hand.
    if new_hand[letter] < 0:
        return False
# If we've gotten to here, we must be able to make the word from this hand.
# Set self.hand to the new, updated hand and return True.
self.hand = new_hand
return True
```

✖ Incorrect

Automatically Generating Problems

“I want to learn about Lists, Append, Slicing”

```
def everyOther(l1,l2):
  x=l1[:__]
  y=l2[:__]
  z = __.append(y)
  return __
```

Python → Constraints

```python
def everyOther(l1,l2):
  x=l1[:2]
  y=l2[:2]
  z = x.append(y)
  return z
```

Python Equations:

1. Define Meaning of variables
2. Define operations/functions

....

....

....

....

Algorithm for Simpler Domain

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

- Easier to Encode as constraints
- General Algorithm for many domains

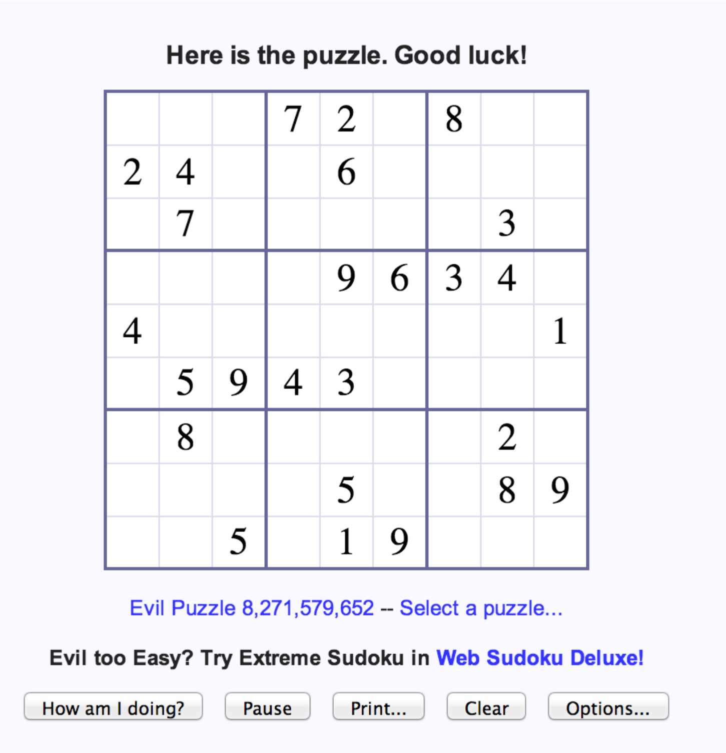
Algorithm for Simpler Domain

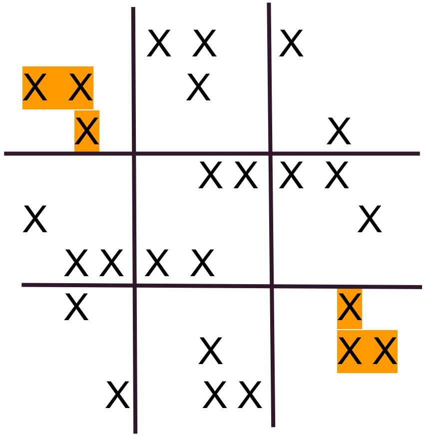
8			7	1	5			4
		5	3		6	7		
3		6	4		8	9		1
	6			5			4	
			8		7			
	5			4			9	
6		9	5		3	4		2
		4	9		2	5		
5			1	6	4			9

Sudoku Constraints:

1. 9x9 square, 81 integers
2. All 81 integers are between 1 and 9
3. Values in row, column, and 3x3 subgrid are distinct

Web Sudoku

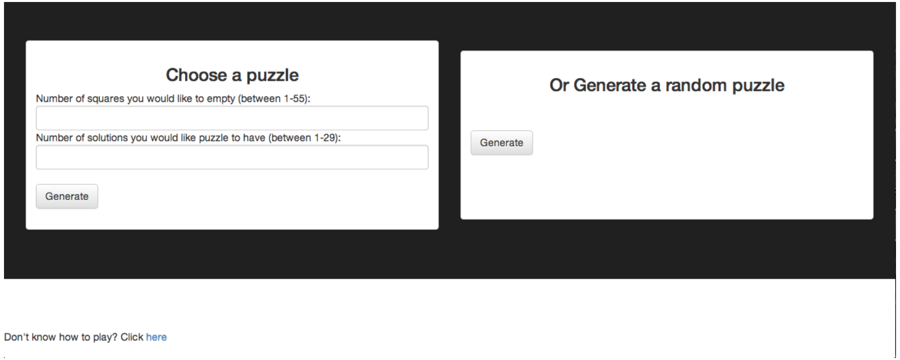


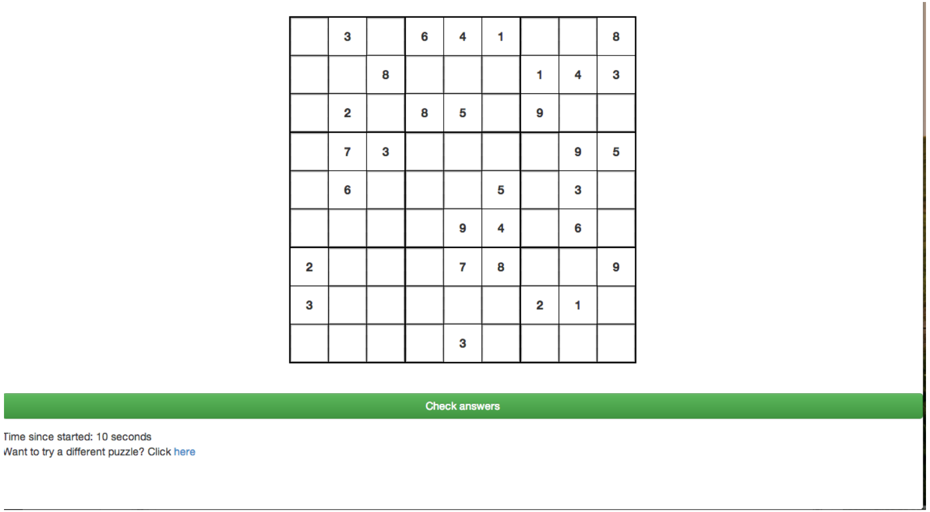


http://www.websudoku.com/

Our Website

- Generates more interesting puzzles
- Has a helpful checker that points to incorrect squares





How was the website made

- Generate around 250,000 puzzles
- Store them in a database with their solutions
- Pick a puzzle depending on user's request (number of squares emptied and number of solutions)
- Check user's filled out board against to solution to find the exact square where the user is incorrect

Breaking Down the Problem

- Automatically generate puzzles of different complexities
- Three main parts to this problem
  - **1. Puzzle:** define what a puzzle means
  - **2. Different Complexity**
  - **3. Automated Generation**

1. Defining the Puzzles

z3 Constraint Solver:

```
X = [[Int('x%d%d' % (i,j)) for i in range
(9)] for j in range(9)]

valid_values = [And ( X[i][j] >= 1, X[i]
[j] <= 9) for i in range(9) for j in range
(9)]
```

1. Define 81 integer values

1. Defining the Puzzles

z3 Constraint Solver:

Each row contains digits 1-9:

```
row_distinct = [Distinct(X[i]) for i in range(9)]
```

Each column contains digits 1-9:

```
cols_distinct = [Distinct([X[i][j] for i in range(9)]) for j in range(9)]
```

Each 3 X 3 square contains digits 1-9:

```
three_by_three_distinct = [ Distinct([X[3*k + i][3*l + j] for i in range(3) for j in range(3)]) for k in range(3) for l in range(3)]
```

1. Define 81 integer values
2. Add Sudoku constraints

1. Defining the Puzzles

z3 Constraint Solver:

```
already_set = [X[i][j] == board[i][j] if
board[i][j] != 0 for i in range(9) for j in
range(9)]
```

1. Define 81 integer values
2. Add Sudoku constraints
3. Encode partially filled Sudoku

1. Defining the Puzzles

z3 Constraint Solver:

```
sudoku_constraint = valid_values +
row_distinct + cols_distinct +
three_by_three_distinct + already_set
```

1. Define 81 integer values
2. Add Sudoku constraints
3. Encode partially filled Sudoku
4. Combine all constraints to form complete set of Sudoku constraints

2. Defining Complexity

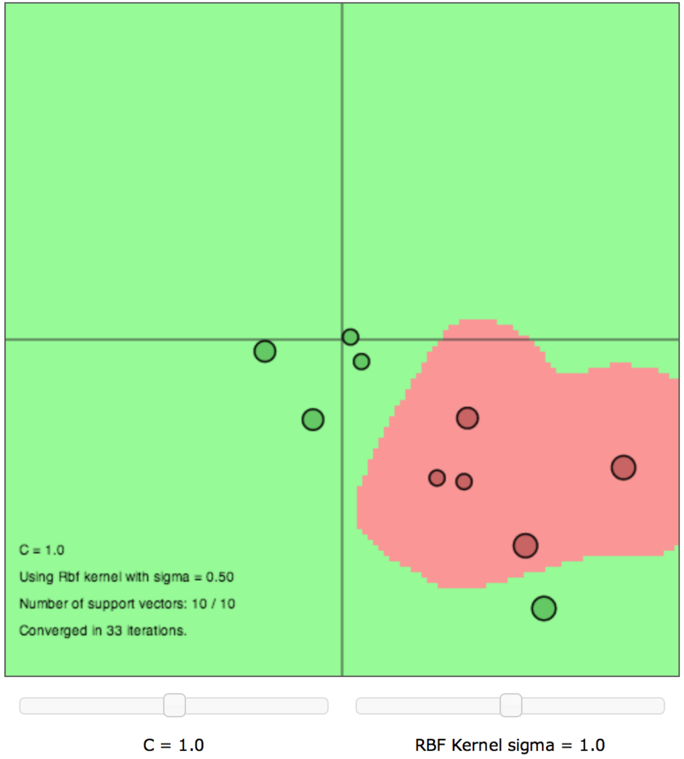
Web Sudoku FAQ:

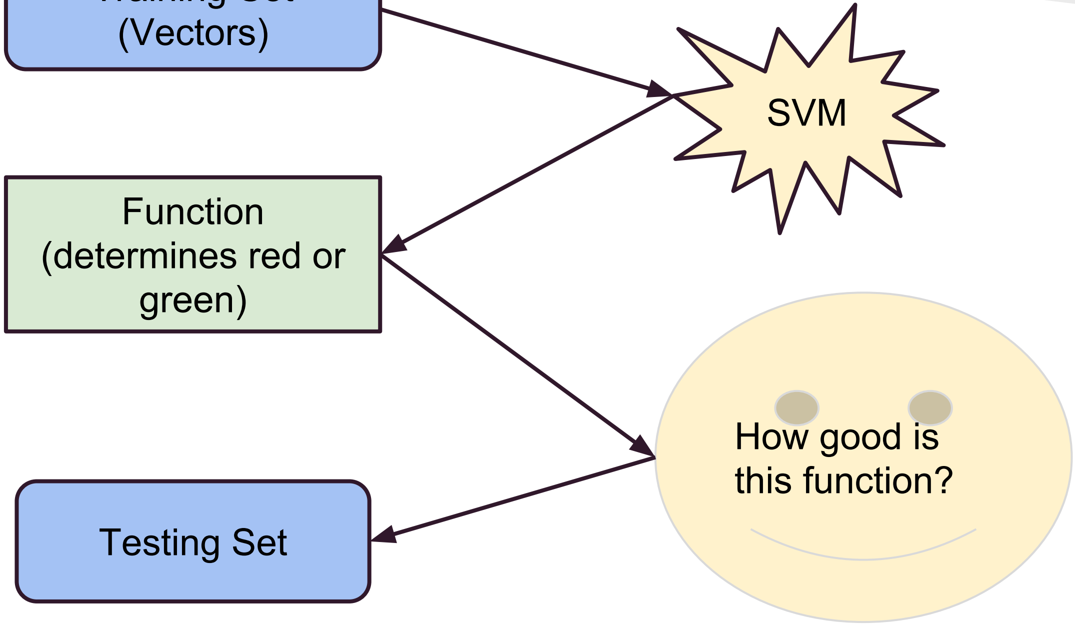
How do you grade the level of the puzzles?

Every puzzle is graded based on the **depth of logical reasoning required**. Our Sudokus never require 'brute force' or 'trial and error' methods, which are easy for computers but impossible for humans working with pen and paper.

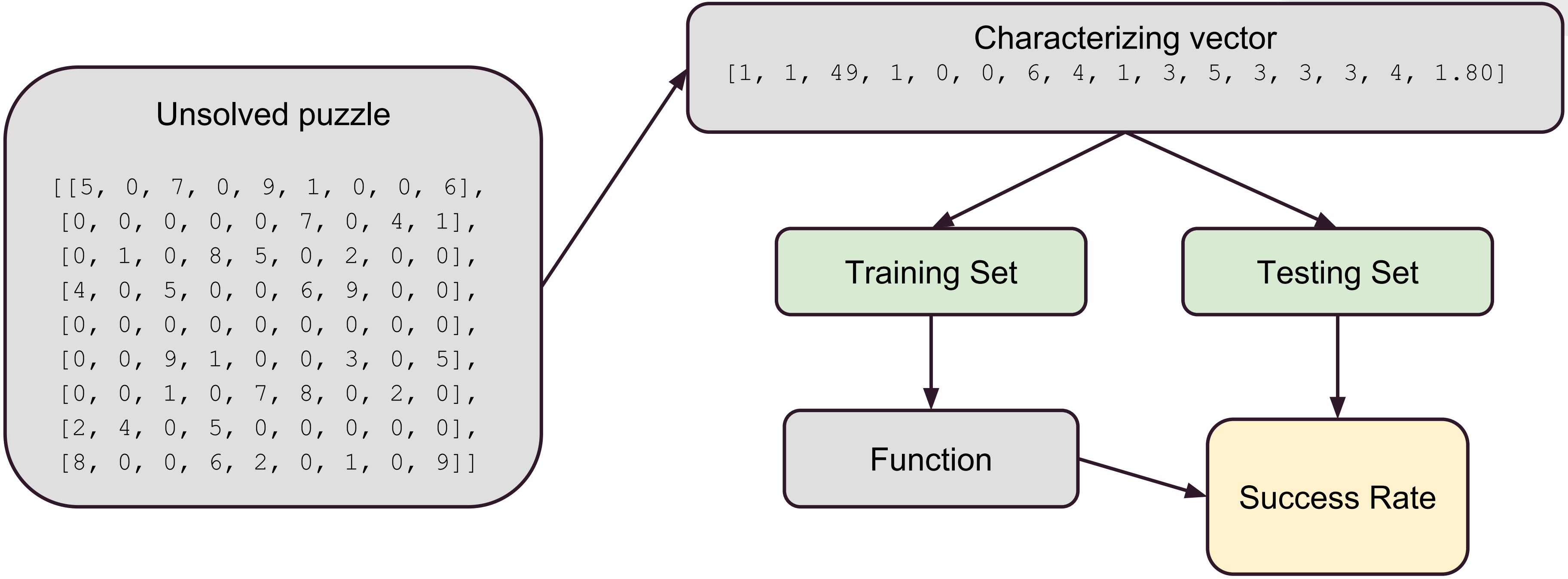
We took a machine learning based approach.

Support Vector Machines (SVM)





2. Defining Complexity



2. Defining Complexity

Characterizing vector

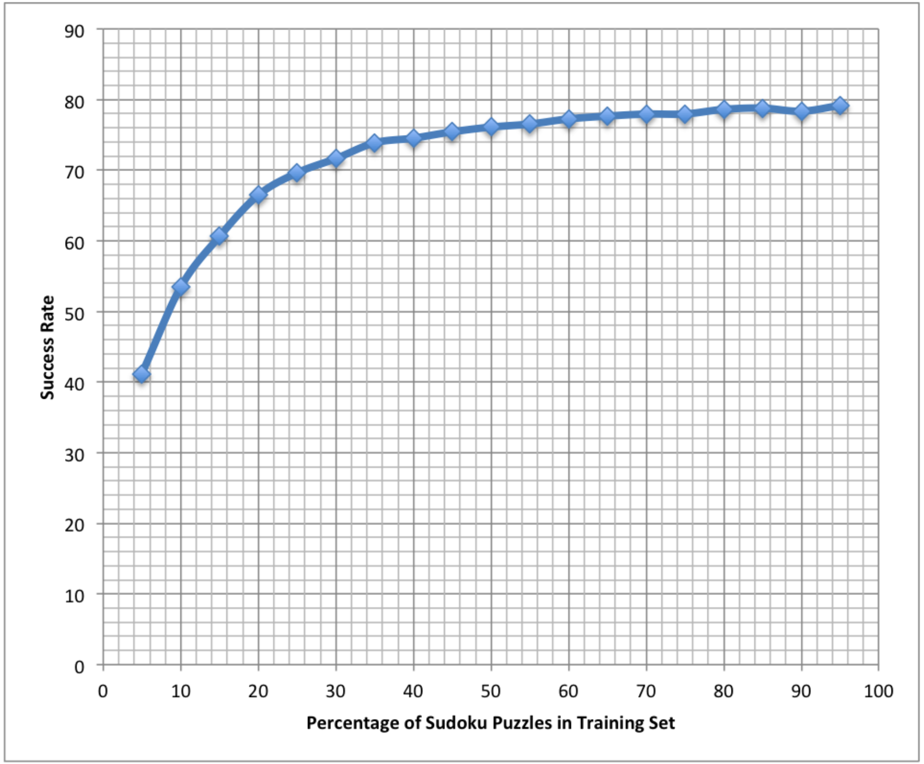
`[1, 1, 49, 1, 0, 0, 6, 4, 1, 3, 5, 3, 3, 3, 4, 1.80]`

1. Difficulty (1, 2, 3, or 4)
2. Number of solutions (always 1 for puzzles from Web Sudoku)
3. Number of empty squares
4. Density of rows
5. Density of columns
6. Density of 3x3 sub-grids

7 – 15. Number of occurrences of each digit

16. Standard deviation of number of occurrences

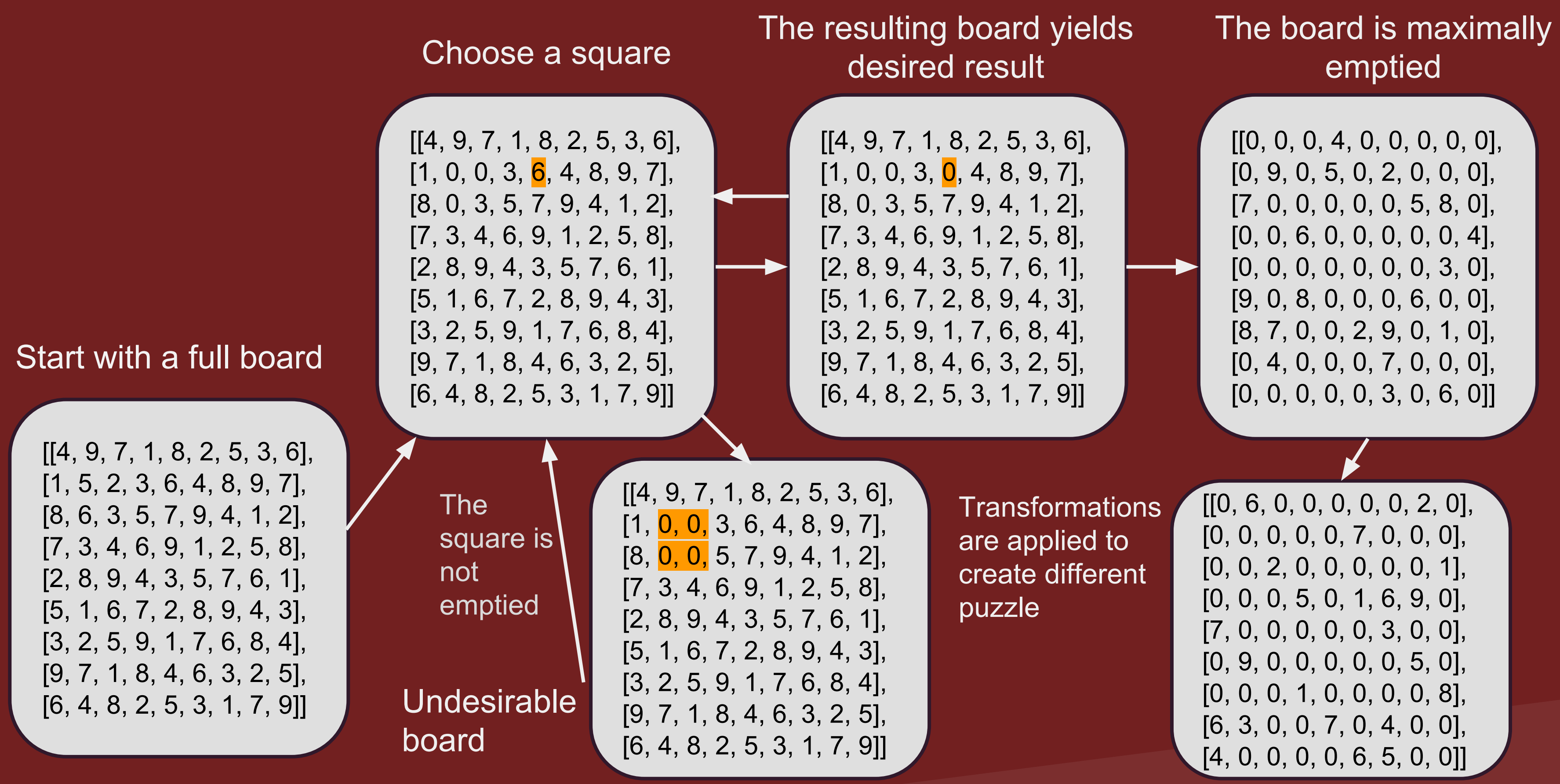
2. Defining Complexity



- 80% Success Rate
- Good indicator of difficulty

3. The Algorithm

- ★ Generate a solution
- ★ Strategically empty elements from the solutions
- ★ Apply a series of transformations to the emptied solution



Visual Representation of the algorithm on a Sudoku Puzzle

Step 1: Generate a full puzzle

- Using z3 constraint solver, generate a full puzzle
- Perform transformations on this puzzle to create more

```
[[4, 9, 7, 1, 8, 2, 5, 3, 6],
[1, 5, 2, 3, 6, 4, 8, 9, 7],
[8, 6, 3, 5, 7, 9, 4, 1, 2],
[7, 3, 4, 6, 9, 1, 2, 5, 8],
[2, 8, 9, 4, 3, 5, 7, 6, 1],
[5, 1, 6, 7, 2, 8, 9, 4, 3],
[3, 2, 5, 9, 1, 7, 6, 8, 4],
[9, 7, 1, 8, 4, 6, 3, 2, 5],
[6, 4, 8, 2, 5, 3, 1, 7, 9]]
```

Step 2: Select a square to empty

- Pick a random row
- Find the percentage of squares that are full in that row
- Generate a random decimal between 0 and 1
- If this decimal is less than the percentage, keep the row
- If the decimal is greater than the percentage, try again with a new row and a new decimal
- Go through same process to generate the column

```
[[4, 9, 7, 1, 8, 2, 5, 3, 6],
[1, 0, 0, 3, 6, 4, 8, 9, 7],
[8, 0, 3, 5, 7, 9, 4, 1, 2],
[7, 3, 4, 6, 9, 1, 2, 5, 8],
[2, 8, 9, 4, 3, 5, 7, 6, 1],
[5, 1, 6, 7, 2, 8, 9, 4, 3],
[3, 2, 5, 9, 1, 7, 6, 8, 4],
[9, 7, 1, 8, 4, 6, 3, 2, 5],
[6, 4, 8, 2, 5, 3, 1, 7, 9]]
```

Step 3: What to do with a selected square

Desirable result: puzzle that has a number of solutions < K
We looked at many different values of K, but had a focus on when K=2

- If the puzzle yields a desirable result, continue emptying squares

```
[[4, 9, 7, 1, 8, 2, 5, 3, 6],
[1, 0, 0, 3, 0, 4, 8, 9, 7],
[8, 0, 3, 5, 7, 9, 4, 1, 2],
[7, 3, 4, 6, 9, 1, 2, 5, 8],
[2, 8, 9, 4, 3, 5, 7, 6, 1],
[5, 1, 6, 7, 2, 8, 9, 4, 3],
[3, 2, 5, 9, 1, 7, 6, 8, 4],
[9, 7, 1, 8, 4, 6, 3, 2, 5],
[6, 4, 8, 2, 5, 3, 1, 7, 9]]
```

- If the puzzle yields an undesirable result, do not empty the square and pick another square to empty

```
[[4, 9, 7, 1, 8, 2, 5, 3, 6],
[1, 0, 0, 3, 6, 4, 8, 9, 7],
[8, 0, 0, 5, 7, 9, 4, 1, 2],
[7, 3, 4, 6, 9, 1, 2, 5, 8],
[2, 8, 9, 4, 3, 5, 7, 6, 1],
[5, 1, 6, 7, 2, 8, 9, 4, 3],
[3, 2, 5, 9, 1, 7, 6, 8, 4],
[9, 7, 1, 8, 4, 6, 3, 2, 5],
[6, 4, 8, 2, 5, 3, 1, 7, 9]]
```

Generating Full Boards

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

1. Switch Columns

Generating Full Boards

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

1. Switch Columns
2. Switch Rows

Generating Full Boards

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

1. Switch Columns
2. Switch Rows
3. Switch Bands

Generating Full Boards

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

1. Switch Columns
2. Switch Rows
3. Switch Bands
4. Switch Stacks

Generating Full Boards

4	5	1	2	9	6	3	7	8
3	2	9	8	5	7	4	6	1
8	6	7	1	4	3	5	9	2
7	4	5	3	2	9	8	1	6
6	9	8	5	[illegible]	4	7	2	3
2	1	3	7	6	8	9	5	4
5	3	6	4	7	1	2	8	9
9	7	4	6	8	2	1	3	5
1	8	2	9	3	5	6	4	7

1. Switch Columns
2. Switch Rows
3. Switch Bands
4. Switch Stacks
5. Reflect

Generating Full Boards

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

1. Switch Columns
2. Switch Rows
3. Switch Bands
4. Switch Stacks
5. Reflect
6. Rotate

Generating Full Boards

[1, 2, 3, 4, 5, 6, 7, 8, 9]

[4, 8, 9, 1, 3, 2, 5, 7, 6]

1. Switch Columns
2. Switch Rows
3. Switch Bands
4. Switch Stacks
5. Reflect
6. Rotate
7. Permute digits

Generating Full Boards

Pros:
- Very fast
- Works with 12x12, 15x15, 16x16, etc. boards

Con: Only $3x10^6$ boards

1. Switch Columns
2. Switch Rows
3. Switch Bands
4. Switch Stacks
5. Reflect
6. Rotate
7. Permute digits

Compatibility with other problems

16 X 16 Puzzle

[[13, 8, 4, 2, 16, 6, 10, 12, 9, 11, 7, 3, 15, 5, 14, 1],
[9, 1, 5, 12, 13, 15, 8, 3, 4, 6, 14, 10, 7, 2, 11, 16],
[6, 14, 11, 7, 9, 5, 2, 4, 15, 16, 12, 1, 13, 3, 8, 10],
[15, 16, 10, 3, 7, 14, 1, 11, 2, 13, 8, 5, 6, 12, 4, 9],
[7, 12, 2, 13, 8, 3, 6, 9, 16, 1, 15, 4, 11, 10, 5, 14],
[3, 6, 1, 10, 14, 4, 16, 7, 5, 12, 9, 11, 2, 15, 13, 8],
[11, 4, 15, 8, 12, 1, 5, 13, 10, 14, 6, 2, 16, 9, 3, 7],
[14, 5, 16, 9, 10, 2, 11, 15, 13, 7, 3, 8, 12, 6, 1, 4],
[16, 2, 14, 5, 1, 12, 13, 8, 7, 9, 10, 6, 4, 11, 15, 3],
[4, 9, 13, 15, 5, 11, 3, 6, 8, 2, 1, 16, 10, 14, 7, 12],
[12, 7, 8, 6, 15, 10, 9, 14, 3, 4, 11, 13, 5, 1, 16, 2],
[10, 11, 3, 1, 4, 16, 7, 2, 14, 15, 5, 12, 8, 13, 9, 6],
[8, 3, 6, 16, 11, 13, 12, 5, 1, 10, 4, 14, 9, 7, 2, 15],
[2, 15, 12, 14, 3, 7, 4, 16, 6, 5, 13, 9, 1, 8, 10, 11],
[1, 13, 9, 11, 2, 8, 15, 10, 12, 3, 16, 7, 14, 4, 6, 5],
[5, 10, 7, 4, 6, 9, 14, 1, 11, 8, 2, 15, 3, 16, 12, 13]]

25 X 25 Puzzle

[[4, 21, 7, 18, 13, 3, 6, 15, 9, 20, 24, 12, 16, 25, 2, 22, 11, 17, 14, 5, 10, 1, 19, 8, 23],
[5, 9, 19, 1, 12, 14, 18, 8, 24, 23, 11, 22, 17, 15, 10, 21, 6, 7, 4, 3, 25, 13, 2, 20, 16],
[3, 16, 22, 8, 23, 17, 1, 4, 7, 25, 19, 13, 6, 18, 14, 10, 24, 20, 15, 2, 11, 5, 9, 12, 21],
[2, 15, 24, 11, 10, 13, 21, 16, 5, 19, 3, 8, 20, 23, 7, 18, 25, 9, 12, 1, 14, 4, 17, 6, 22],
[20, 25, 6, 14, 17, 12, 22, 10, 11, 2, 1, 21, 4, 5, 9, 16, 19, 23, 8, 13, 3, 7, 24, 18, 15],
[14, 18, 8, 6, 16, 20, 17, 7, 23, 13, 15, 11, 3, 4, 21, 1, 12, 25, 24, 19, 9, 2, 22, 10, 5],
[25, 22, 15, 2, 7, 24, 3, 21, 18, 10, 8, 6, 23, 1, 19, 14, 5, 4, 9, 11, 13, 17, 12, 16, 20],
[10, 17, 13, 9, 3, 22, 19, 11, 14, 5, 7, 24, 18, 16, 12, 6, 15, 2, 20, 23, 4, 25, 1, 21, 8],
[19, 24, 21, 4, 11, 25, 2, 12, 15, 1, 20, 9, 22, 14, 5, 13, 17, 8, 10, 16, 18, 23, 6, 3, 7],
[1, 5, 23, 12, 20, 8, 4, 9, 16, 6, 10, 17, 25, 2, 13, 7, 22, 3, 18, 21, 19, 15, 11, 14, 24],
[7, 4, 16, 15, 6, 9, 24, 2, 20, 22, 17, 5, 12, 8, 18, 19, 21, 13, 3, 10, 23, 11, 25, 1, 14],
[23, 13, 2, 19, 21, 4, 5, 18, 10, 11, 22, 14, 24, 3, 25, 9, 7, 6, 1, 20, 8, 12, 16, 15, 17],
[9, 3, 10, 17, 14, 23, 25, 6, 8, 15, 13, 7, 1, 20, 16, 24, 4, 5, 11, 12, 22, 18, 21, 19, 2],
[18, 20, 12, 24, 25, 21, 14, 1, 13, 16, 23, 10, 11, 19, 4, 17, 8, 22, 2, 15, 5, 3, 7, 9, 6],
[11, 8, 1, 22, 5, 19, 12, 3, 17, 7, 6, 2, 21, 9, 15, 23, 14, 16, 25, 18, 20, 24, 13, 4, 10],
[17, 1, 18, 10, 8, 15, 9, 5, 12, 14, 2, 20, 13, 11, 6, 3, 16, 21, 23, 25, 7, 22, 4, 24, 19],
[13, 2, 3, 20, 19, 16, 23, 24, 1, 4, 14, 15, 8, 10, 22, 5, 9, 12, 7, 17, 21, 6, 18, 11, 25],
[6, 11, 14, 7, 24, 10, 20, 25, 22, 18, 16, 23, 5, 21, 1, 15, 2, 19, 13, 4, 12, 8, 3, 17, 9],
[15, 23, 4, 5, 9, 6, 11, 17, 19, 21, 18, 25, 7, 12, 3, 8, 20, 1, 22, 24, 16, 14, 10, 2, 13],
[21, 12, 25, 16, 22, 7, 8, 13, 2, 3, 4, 19, 9, 24, 17, 11, 10, 18, 6, 14, 15, 20, 23, 5, 1],
[24, 19, 11, 13, 4, 2, 16, 14, 6, 9, 12, 18, 10, 22, 8, 20, 23, 15, 17, 7, 1, 21, 5, 25, 3],
[12, 7, 20, 25, 1, 11, 15, 22, 21, 17, 5, 16, 2, 13, 24, 4, 3, 10, 19, 8, 6, 9, 14, 23, 18],
[16, 10, 9, 21, 2, 1, 7, 23, 4, 8, 25, 3, 14, 6, 20, 12, 18, 24, 5, 22, 17, 19, 15, 13, 11],
[22, 14, 5, 3, 15, 18, 10, 20, 25, 12, 9, 1, 19, 17, 23, 2, 13, 11, 21, 6, 24, 16, 8, 7, 4],
[8, 6, 17, 23, 18, 5, 13, 19, 3, 24, 21, 4, 15, 7, 11, 25, 1, 14, 16, 9, 2, 10, 20, 22, 12]]

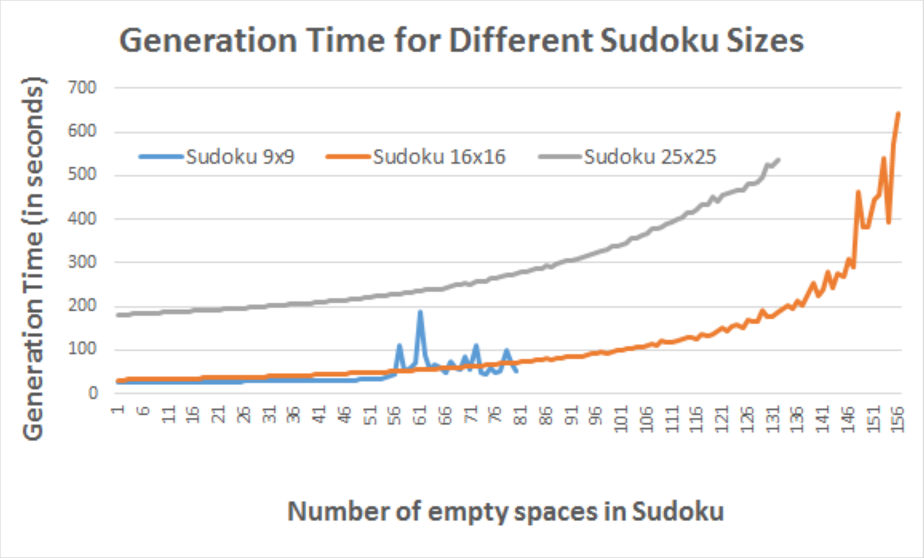
Minimal Changes in Code

```
def createSudoku(board, n):
    X = [[Int('x%dd%dd' % (i,j)) for i in range(n)] for j in range(n)]
    valid_values = [And ( X[i][j] >= 1, X[i][j] <= n) for i in range(n) for j in range(n)]
    # Every row should be disntinct
    row_distinct = [Distinct(X[i]) for i in range(n)]
    # Every column should be disntinct
    cols_distinct = [Distinct([X[i][j] for i in range(n)]) for j in range(n)]
    # Every 3 x 3 square should be disntinct
    three_by_three_distinct = [ Distinct([X[n**(1/2)*k + i][n**(1/2)*l + j] for i in range(n**(1/2)) for j in range(n**(1/2))]) for k in range(n**(1/2)) for l in range(n**(1/2))]
    # There are values already set in the board, which we need to take into account
    s = Solver()
    s.add(valid_values + row_distinct + cols_distinct + three_by_three_distinct)
    if s.check() == sat:
        m = s.model()
        r = [ [ m.evaluate(X[i][j]) for j in range(n) ] for i in range(n) ]
    return r

board = createSudoku([], 9)
```

Only have to change n to generate new Sudoku puzzles of different complexity

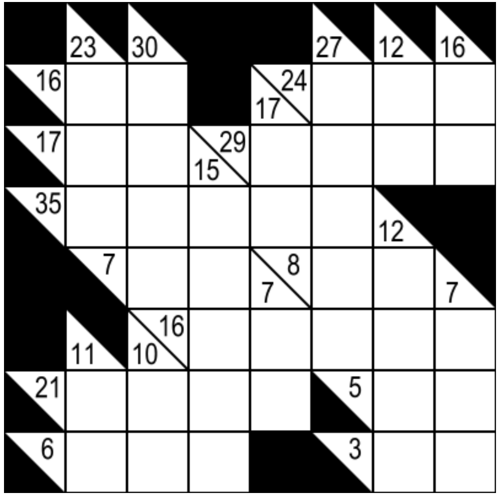
Experimental Results

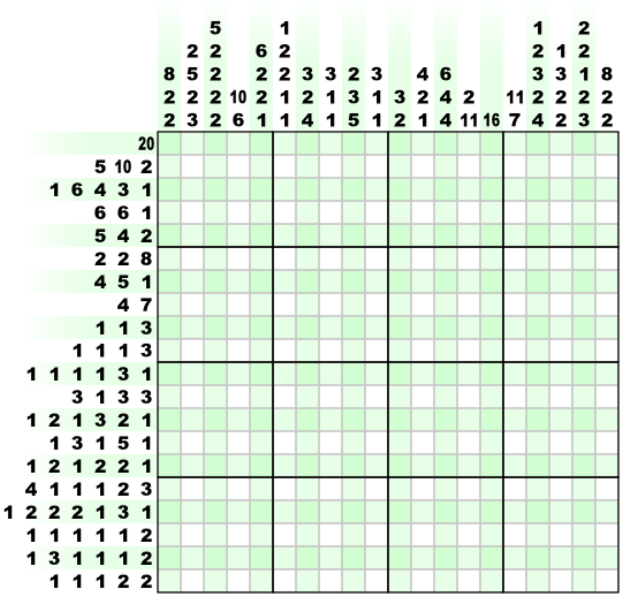


Size	Max Empty Squares	% Empty Squares
9x9	60	74%
16x16	163	64%
25x25	281	45%

Future Work

- Generate more constraint-based puzzles





Future Work

- Generate more constraint-based puzzles
- Extend algorithm to automatically generate Python programming problems

```python
def everyOther(l1,l2):
  x=l1[:2]
  y=l2[:2]
  z = x.append(y)
  return z
```

Sketch

```python
def everyOther(l1,l2):
  x=l1[:__]
  y=l2[:__]
  z = __.append(y)
  return __
```

Future Work

- Generate more constraint-based puzzles
- Extend algorithm to automatically generate Python programming problems
- Generate math problems (algebra, trigonometry, geometry, etc.)

Special Thanks to...

- Mentor: Rishabh Singh
- Professor: Armando Solar-Lezama
- The MIT-PRIMES Program
- Our parents