Wilfrid Laurier University

Assignment 2

Sudoku CSP

October 29, 2022

Nahor Yirgaalem - 190775540 - yirg5540@mylaurier.ca

Shaheer Khan - 190693830 - khan3830@mylaurier.ca

Duong Truong - 190695030 - truo9503@mylaurier.ca

Sameer Mian - 190366140 - mian6140@mylaurier.ca

Memet Rusidovski - 130951550 - rusi1550@mylaurier.ca

CP468: Artificial Intelligence

Dr. Illias kotsireas

```
7 8 5  | 4 3 9  | 1 2 6
6 1 2  | 8 7 5  | 3 4 9
4 9 3  | 6 2 1  | 5 7 8
- - - - - - - - - - - - - - -
8 5 7  | 9 4 3  | 2 6 1
2 6 1  | 7 5 8  | 9 3 4
9 3 4  | 1 6 2  | 7 8 5
- - - - - - - - - - - - - - -
5 7 8  | 3 9 4  | 6 1 2
1 2 6  | 5 8 7  | 4 9 3
3 4 9  | 2 1 6  | 8 5 7
Arc Queue Size:  0
SOLVED
a0 - x
a1 - x
a2 - [5]
a3 - x
a4 - [3]
a5 - [9]
a6 - x
a7 - x
a8 - [6]
b0 - x
```

```
12
11
10
9
8
7
6
5
4
3
2
1
0
7 8 5  | 4 3 9  | 1 2 6
6 1 2  | 8 7 5  | 3 4 9
4 9 3  | 6 2 1  | 5 7 8
- - - - - - - - - - - - - - - -
8 5 7  | 9 4 3  | 2 6 1
2 6 1  | 7 5 8  | 9 3 4
9 3 4  | 1 6 2  | 7 8 5
- - - - - - - - - - - - - - - -
5 7 8  | 3 9 4  | 6 1 2
1 2 6  | 5 8 7  | 4 9 3
```

```
import ast
from func import *
from queue import Queue

"""
-------------------------------------------------------------------------------------------
File:    sudoku.py
Project: AI_Project_Repo
Purpose:
===========================================================================================

Program Description:
  This program solves a sudoku puzzle giving in the format
  of a 2d array. The program reads the first puzzle from the
  txt file and attempts to solve it.
-------------------------------------------------------------------------------------------
Group:   14
Email:   rusi1550@mylaurier.ca
Version  2022-11-09
-------------------------------------
"""

#Default Sudoku Board
board = [
    [7, 8, 0, 4, 0, 0, 1, 2, 0],
    [6, 0, 0, 0, 7, 5, 0, 0, 9],
    [0, 0, 0, 6, 0, 1, 0, 7, 8],
    [0, 0, 7, 0, 4, 0, 2, 6, 0],
    [0, 0, 1, 0, 5, 0, 9, 3, 0],
    [9, 0, 4, 0, 6, 0, 0, 0, 5],
    [0, 7, 0, 3, 0, 0, 0, 1, 2],
    [1, 2, 0, 0, 0, 7, 4, 0, 0],
    [0, 4, 9, 2, 0, 6, 0, 0, 7]
]


with open('/Users/schoolaccount/Documents/GitHub/AI_Project_Repo/Assignment_2/sudoku.txt') as
f:
    temp = []
    x = f.readlines()
    for lines in x:
        if lines != '\n' and lines[0] != '#':
            w = ast.literal_eval(lines)
            w = [int(x) for x in w]
            temp.append(w)
        else:
            break

    print(print_board(temp))
    board = temp


# Variables - All zero's, Constraints - Rules of Game, Domains - All possible scenario's
arc = Queue()
domain = {}

# Populate domain and arc queue
createDomain(board, domain)
#printDomain(domain)
createArcQueue(domain, arc)


AC3(arc, domain, board)

# If the queue is empty the puzzle is solved
if arc.qsize() == 0:
    print_board(board)
```

```
    print("Arc Queue Size: ",arc.qsize())
    print("SOLVED")
else:
    # Finish solving the board
    backtrack(board, 0, 0)
    printDomain(domain)

    print_board(board)

    print("WAS NOT SOLVED - Backtracking...")

#printArc(arc)
printDomain(domain)
```

```python
from copy import deepcopy
from queue import Queue

"""
-----------------------------------------------------------------------------------------
File:    func.py
Project: AI_Project_Repo
Purpose:
=========================================================================================

Program Description:
  This is a file for our functions. This helps keep the main file
  readable.
-----------------------------------------------------------------------------------------
Group:   14
Email:   rusi1550@mylaurier.ca
Version  2022-11-09
-------------------------------------
"""

abc = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8}

# Help Print the board
def print_board(b):
    for i in range(len(b)):
        if i % 3 == 0 and i != 0:
            print("- - - - - - - - - - - - ")

        for j in range(len(b[0])):
            if j % 3 == 0 and j != 0:
                print(" | ", end="")

            if j == 8:
                print(b[i][j])
            else:
                print(str(b[i][j]) + " ", end="")

#Create a domain for each empty cell
def createDomain(board, q):
    abc = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

    for x, letter in zip(board, abc):
        count = 0
        for y in x:
            if y == 0:
                q[f"{letter}{count}"] = [1, 2, 3, 4, 5, 6, 7, 8, 9]
            else:
                q[f"{letter}{count}"] = "x"
            count += 1

# This function creates a queue of arcs the are
# required for the sudoku puzzle
def createArcQueue(domain, arc):
    abc = {'a': 0, 'b': 1, 'c':2, 'd':3, 'e':4, 'f':5, 'g':6, 'h':7, 'i':8}

    for i in domain:
        # Check for arcs only in empty squares
        if domain[i] != 'x':
            # Row Consistency
            row = abc[i[0]]
            for x in range(9):
                if x != int(i[1]):
                    arc.put([i, (row, x)])

            # Colum Consistency
            col = int(i[1])
            for y in range(9):
```

```
                if y != abc[i[0]]:
                    arc.put([i, (y, col)])

            # Box Consistency
            quadY = abc[i[0]] - abc[i[0]] % 3
            quadX = int(i[1]) - int(i[1]) % 3

            for y in range(3):
                for x in range(3):
                    if (quadX + x) != int(i[1]) or (quadY + y) != abc[i[0]]:
                        arc.put([i, (quadY + y, quadX + x)])

    return

# the AC3 algorithm
def AC3(arc, domain, board):
    B = True
    c = 0
    isTrue = True

    while not arc.empty() and c < 10000 and isTrue:
        isTrue = revise(arc, domain, board)
        c += 1

    return B


def revise(arc, domain, board):
    B = True
    temp = arc.get()

    # Display the queue size in terminal
    print(arc.qsize())

    # If the cell number is in the domain of the empty cell remove it from
    # it's domain
    if board[temp[1][0]][temp[1][1]] in domain[temp[0]]:
        domain[temp[0]].remove(board[temp[1][0]][temp[1][1]])


    # If the empty cell has a domain of zero that means the board is not solvable
    # and return false
    if len(domain[temp[0]]) == 0:
        B = False

    # If domain is one then the cell is solved
    if len(domain[temp[0]]) == 1:
        s = temp[0]
        board[abc[s[0]]][int(s[1])] = domain[temp[0]][0]

    # If the domain is 2 or more then an arc between the points is still needed
    if len(domain[temp[0]]) >= 2:
        arc.put(temp)

    return B

def backtrack(board, row, col):
    if (row == 8 and col == 9):
        return True

    if col == 9:
        row += 1
        col = 0

    if board[row][col] > 0:
        return backtrack(board, row, col + 1)
```

```
    for num in range(1, 10):
        if check(board, row, col, num):
            board[row][col] = num

            if backtrack(board, row, col + 1):
                return True

        board[row][col] = 0
    return False


def check(board, row, col, num):
    for x in range(9):
        if board[row][x] == num:
            return False

    for x in range(9):
        if board[x][col] == num:
            return False

    quadX = row - row % 3
    quadY = col - col % 3

    for i in range(3):
        for j in range(3):
            if board[i + quadX][j + quadY] == num:
                return False

    return True


# Some helper print functions mostly used for debugging
def printDomain(domain):
    for i in domain:
        print(i, "-",domain[i])

def printArc(arc):
    tmp = arc
    while not tmp.empty():
        print(tmp.get())
```