CS 4613 **E. K. Wong**
Project 2: Hyper Sudoku **Fall 2020**

Total number of points = 100. You can work on the project yourself or you can work with a partner (a team of two.)

Project Description: Design and implement a program to solve Hyper Sudoku puzzles. Hyper Sudoku differs from the classic Sudoku in that four overlapping regions are defined in addition to the regular regions, as described below. The rules of the game are:

- The game board consists of 9×9 cells divided into 3×3 non-overlapping regions (See Figure 1.) Four additional overlapping regions are defined, as highlighted in green. The game board therefore contains 9 non-overlapping regions and 4 overlapping regions, with each region containing 3×3 cells. Some of the cells already have numbers (1 to 9) assigned to them initially.
- The goal is to find assignments (1 to 9) for the empty cells so that every row, column, non-overlapping region and overlapping region contains all the digits from 1 to 9. Each of the 9 digits, therefore, can only appear once in every row, column, non-overlapping region and overlapping region. Figure 2 shows the solution for the initial game board in Figure 1.

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

Figure 1. Initial game board

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

Figure 2. Solution

As a first step in your program, apply *Forward Checking* to cells that already have a number assigned to them and reduce the domain of their neighbors. If any cell has an empty domain after applying Forward Checking, then the puzzle does not have a solution and the program can stop and exit. Here, you run the Forward Checking algorithm before you run the Backtracking Algorithm for CSPs. Next, use the *Backtracking Algorithm* in Figure 4 below to solve for a solution. Implement the function *SELECT-UNASSIGNED-VARIABLE* by using the *minimum remaining value* heuristic and then the *degree* heuristic. If there are more than one variables left after applying the two heuristics, you can arbitrarily choose a variable to work on next. You do not have to implement the *least constraining value* heuristic in the *ORDER-DOMAIN-VALUES* function; instead, simply order the domain values in increasing order, from 1 to 9. You do not have to implement the INFERENCE function inside the Backtracking Algorithm. (Remember: two or more variables are neighbors if they share a common constraint.)

Input and output files: Your program will read in the initial game board configuration from an input text file and produce an output text file that contains the solution. The input file contains 9

rows (or lines) of integers. Each row contains 9 integers ranging from 0 to 9, separated by blank spaces. Digits 1-9 represent the cell values and 0's represent blank cells. The input file for the initial game board in Figure 1 is shown in Figure 3(a) below. Similarly, the output file contains 9 rows of integers, with each row containing digits ranging from 1 to 9, separated by blank spaces. The output file for the initial game board in Figure 1 is shown in Figure 3(b) below.

Testing your program: Three input test files will be provided on NYU Classes for you to test your program.

Recommended languages: Python, C++/C and Java. If you would like to use a different language, send me an email first.

What to submit: The following files on NYU Classes by the due date. If you have a partner, only one of you needs to submit but put both partners' names on the PDF document. Just submit as separate files (no need to combine files into a single ZIP file.)

1. Your source code file. Put comments in your source code to make it easier for someone else to read your program. Points will be taken off if you do not have comments in your source code.
2. The output text files generated by your program. Name your output files *Output1.txt*, *Output2.txt* and *Output3.txt*.
3. A PDF file that contains the following:
    a. Instructions on how to run your program. If your program requires compilation, instructions on how to compile your program should also be provided.
    b. A description of your formulation of Hyper Sudoku as a constraint satisfaction problem. This includes the set of variables you have defined, the domain for each variable, and the set of constraints you have set up.
    c. Also, copy and paste the output files and your source code onto the PDF file (to make it easier for us to grade your project.) This is in addition to the source code file and output files that you have to submit separately (as described in 1 and 2 above.)

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

(a) Input

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

(b) Output

Figure 3. Input and output files for the initial game board in Figure 1.
Digit 0s in (a) represent blank cells.

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

Figure 4. The Backtracking Algorithm for CSPs.