1. Implement and evaluate a constraint satisfaction problem algorithm.

**Problem description:**

Sudoku is a combinatorial, logic-based, number-placement puzzle. In classic Sudoku, the objective is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 sub-grids that compose the grid contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution (see Figure 1 below). [source: [Sudoku - Wikipedia](https://en.wikipedia.org/wiki/Sudoku)].

|  |  |
| --- | --- |
| 1. unsolved Sudoku puzzle | 1. solved Sudoku puzzle |
| A typical Sudoku puzzle, with nine rows and nine columns that intersect at square spaces. Some of the cells are filled with a number; others are blank cells to be solved. | The previous puzzle, showing its solution. |

*Figure 1: Sudoku puzzle: (a) unsolved, (b) solved* [source: [Sudoku - Wikipedia](https://en.wikipedia.org/wiki/Sudoku)]*.*

task is to implement in Python the following constraint satisfaction problem algorithms

* Brute force (exhaustive) search algorithm,
* Constraint Satisfaction Problem (CSP) back-tracking search,
* CSP with forward-checking and MRV heuristics,

and apply them to solve the puzzle (provided in a CSV file).

* + MODE is mode in which your program should operate
    - 1 – brute force search,
    - 2 – Constraint Satisfaction Problem back-tracking search,
    - 3 – CSP with forward-checking and MRV heuristics,
    - 4 – test if the completed puzzle is correct.
  + FILENAME is the input CSV file name (unsolved or solved sudoku puzzle),

If the number of arguments provided is NOT two (none, one, or more than two) or arguments are invalid (incorrect file, incorrect mode) your program should display the following error message:

Input puzzle:

X,6,X,2,X,4,X,5,X

4,7,X,X,6,X,X,8,3

X,X,5,X,7,X,1,X,X

9,X,X,1,X,3,X,X,2

X,1,2,X,X,X,3,4,X

6,X,X,7,X,9,X,X,8

X,X,6,X,8,X,7,X,X

1,4,X,X,9,X,X,2,5

X,8,X,3,X,5,X,9,X

Solved puzzle:

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

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

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

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

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

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

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

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

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

* Save the solved puzzle to INPUTFILENAME\_SOLUTION.csv file.
* In MODE 4 (test) your program should display the input puzzle along with a message

This is a valid, solved, Sudoku puzzle.

if the solution is correct and

ERROR: This is NOT a solved Sudoku puzzle.

if it is not correct.

**Input data file:**

Your input data file is a single CSV (comma separated values) file containing the Sudoku puzzle grid (see Programming Assignment #02 folder in Blackboard for sample files). The file structure is as follows:

X,6,X,2,X,4,X,5,X

4,7,X,X,6,X,X,8,3

X,X,5,X,7,X,1,X,X

9,X,X,1,X,3,X,X,2

X,1,2,X,X,X,3,4,X

6,X,X,7,X,9,X,X,8

X,X,6,X,8,X,7,X,X

1,4,X,X,9,X,X,2,5

X,8,X,3,X,5,X,9,X

You **CANNOT** modify nor rename input data files. Rows and columns in those files represent individual rows and columns of the puzzle grid as shown on Figure 1. You can assume that file structure is correct without checking it.

CSV file data is either:

* a character X corresponding unassigned (empty) grid cell,
* a positive integer (from the {1, 2, 3, 4, 5, 6, 7, 8, 9} set) corresponding to an assigned grid cell value.

|  |
| --- |
| **Table A** |
| Algorithm | | Number of generated nodes | Average search time in seconds |
| Brute force search | | 241273 | 118.5673 |
| CSP back-tracking | | 7 | 0.00645 |
| CSP with forward-checking and MRV heuristics | | 7 | 0.002223 |

|  |
| --- |
| **Conclusions** |
| In my analysis CSP with forward checking and MRV performs better than the other algorithms, because of exhausting unknowns with maximum constraints with minimum domains. This effectively lead to solving sure shot easy vacancy's and proceeding from there resulted in lesser search time compared CSP back tracking(where the minimal domain parameter is not calculated).  Whereas in brute force, it explores all possible root nodes with unsatisfying Sudoku conditions, and then decides if a root node is solved or not. This makes to explore unnecessary nodes.  Time complexity  Brute force O(9^(n\*n))  CSP back tracking O(n^3) ( ie O(n\*n\*n))  CSP forward checking and MRV O(n^3)   1. Dimension of Sudoku   Same complexity but address the ones with minimum domain(max constraints) |