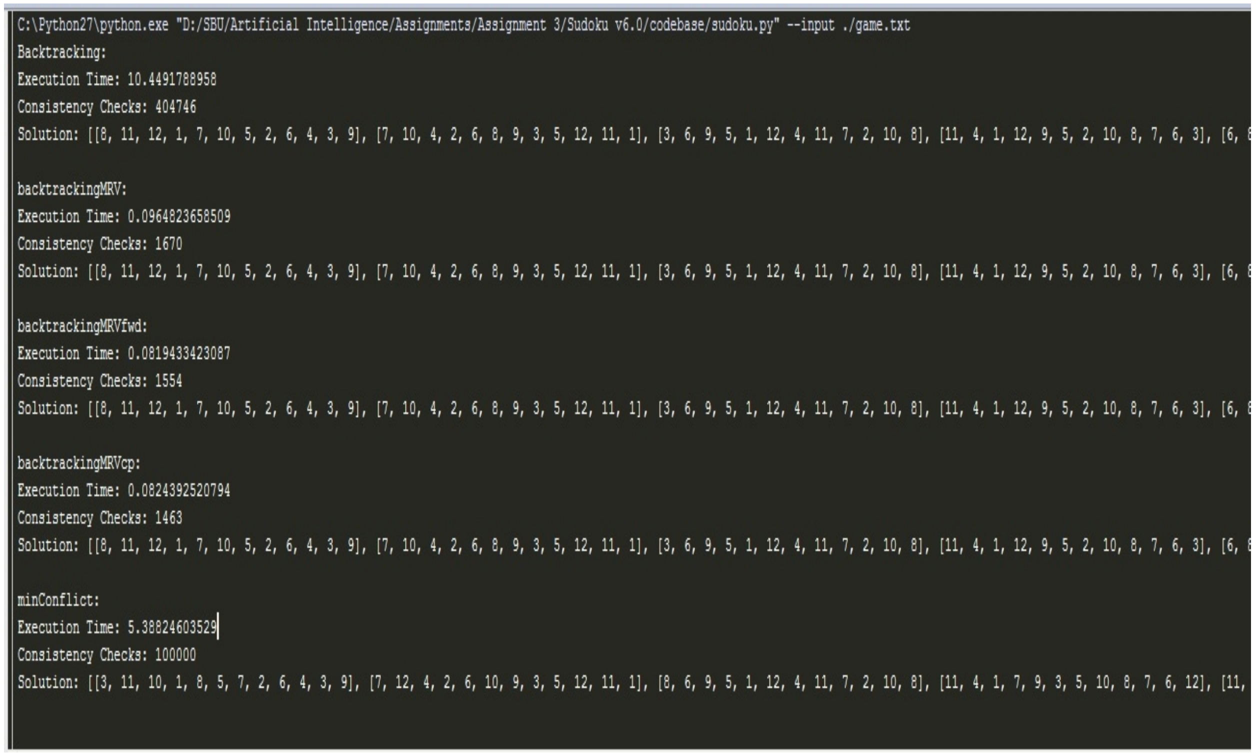
ARTIFICIAL INTELLIGENCE (CSE 537) FALL-2015
ASSIGNMENT 3: SUDOKU

Submitted by:
MOHIT GOYAL (110349552)
DEEPAK GOYAL (110347387)

INTRODUCTION:

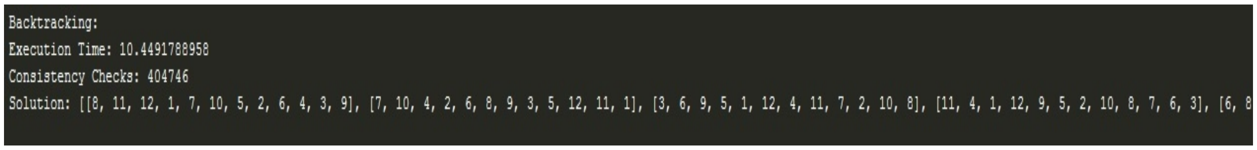
In this assignment, we have implemented backtracking and various constraint satisfaction algorithms to solve the “Sudoku” game more efficiently.

Added SudokuGameSolver.py which implements all the algorithms in SudokuSolver class.



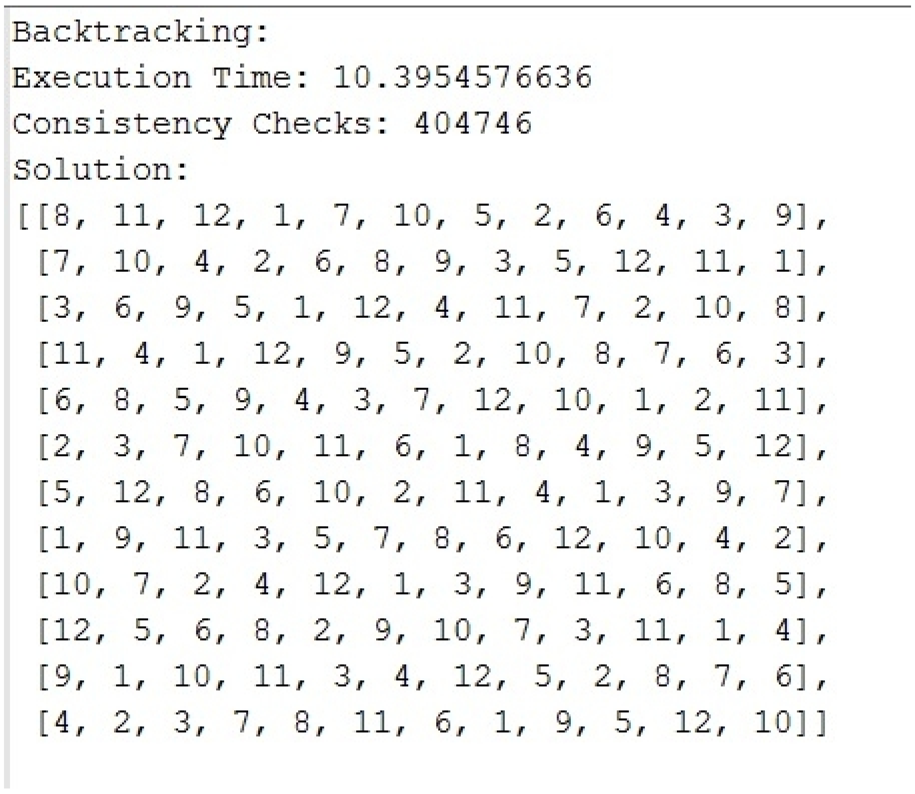
```
C:\Python27\python.exe "D:/SBU/Artificial Intelligence/Assignments/Assignment 3/Sudoku v6.0/codebase/sudoku.py" --input ./game.txt
Backtracking:
Execution Time: 10.4491788958
Consistency Checks: 404746
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 

backtrackingMRV:
Execution Time: 0.0964823658509
Consistency Checks: 1670
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 

backtrackingMRVfwd:
Execution Time: 0.0819433423087
Consistency Checks: 1554
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 

backtrackingMRVcp:
Execution Time: 0.0824392520794
Consistency Checks: 1463
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 

minConflict:
Execution Time: 5.38824603529
Consistency Checks: 100000
Solution: [[3, 11, 10, 1, 8, 5, 7, 2, 6, 4, 3, 9], [7, 12, 4, 2, 6, 10, 9, 3, 5, 12, 11, 1], [8, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 7, 9, 3, 5, 10, 8, 7, 6, 12], [11, 
```

1. **Backtracking:**
   In Backtracking technique, we choose a value from domain for each variable in the CSP problem for Sudoku, check its consistency and recursively call itself.



```
Backtracking:
Execution Time: 10.4491788958
Consistency Checks: 404746
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 8
```

Above output in matrix format:



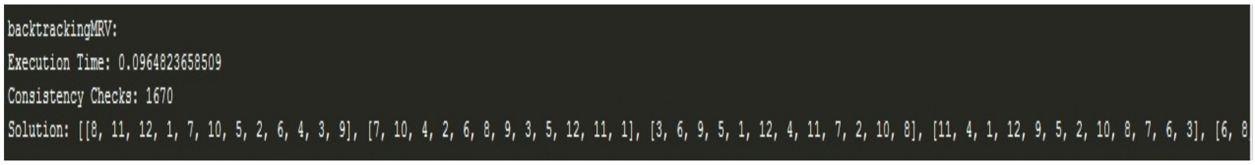
```
Backtracking:
Execution Time: 10.3954576636
Consistency Checks: 404746
Solution:
[[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9],
 [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1],
 [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8],
 [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3],
 [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11],
 [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12],
 [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7],
 [1, 9, 11, 3, 5, 7, 8, 6, 12, 10, 4, 2],
 [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5],
 [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4],
 [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6],
 [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]
```

For small input size n, there is not much difference in the execution time of Backtracking, Backtracking +MRV, Backtracking +MRV+ Forward check, Backtracking +MRV+ Constraint propagation.

2. **Backtracking + MRV heuristic:**
   In this technique, instead of choosing the next unassigned variable in row by row fashion, we select the variable using the MRV heuristic with minimum domain size. It reduces the execution time and consistency check as it prunes the search tree.

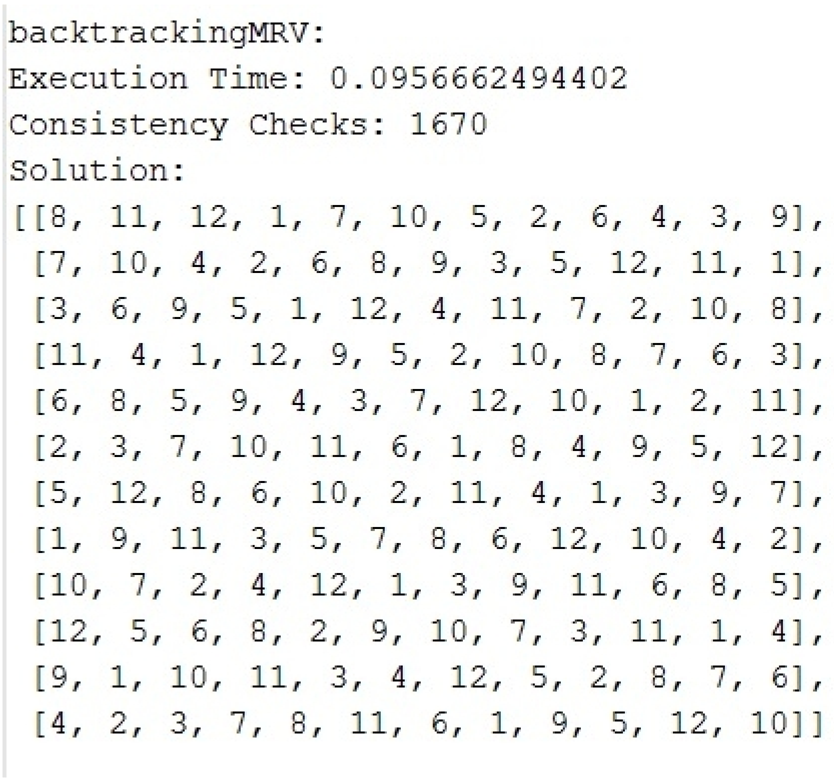
   As we can see from output, for large input size, it performs better than Backtracking approach.

   Output:



```
backtrackingMRV:
Execution Time: 0.0964823658509
Consistency Checks: 1670
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 8
```

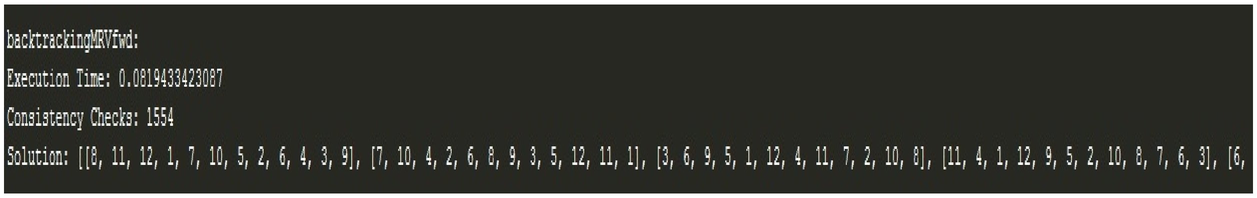
Above output in Matrix Format:



```
backtrackingMRV:
Execution Time: 0.0956662494402
Consistency Checks: 1670
Solution:
[[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9],
 [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1],
 [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8],
 [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3],
 [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11],
 [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12],
 [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7],
 [1, 9, 11, 3, 5, 7, 8, 6, 12, 10, 4, 2],
 [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5],
 [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4],
 [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6],
 [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]
```

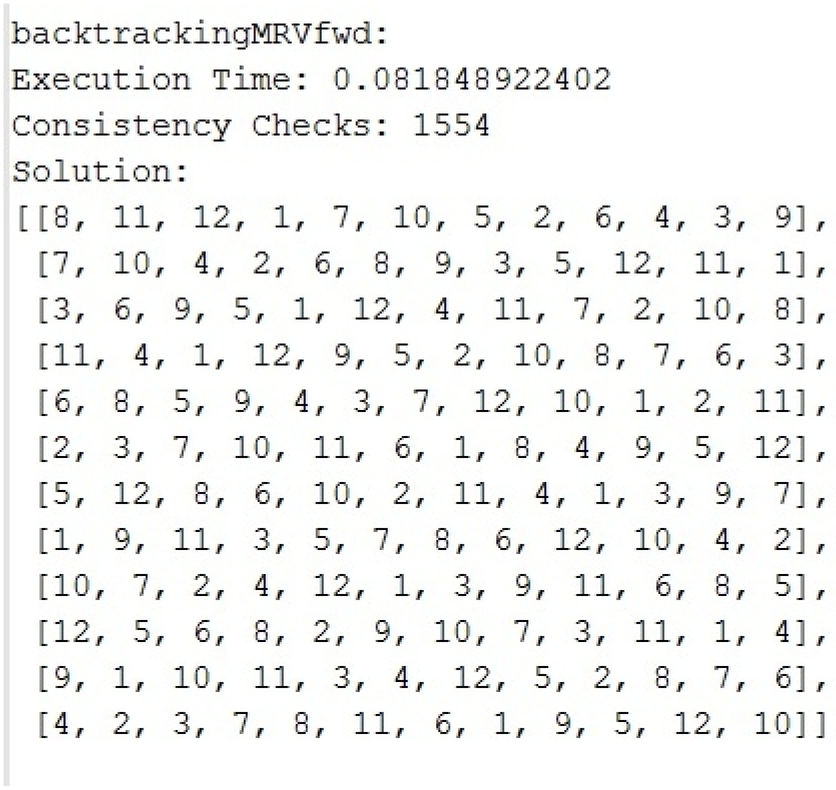
3. **Backtracking + MRV + Forward Checking:**
   MRV heuristic only picks the variable with minimum domain size, but it does not check the state of other constrained variables after applying the AllDiff constraint. It improves the execution time as it prunes the values for variables that will eventually cause a failure in future. For large input size, Forward checking along with MRV heuristic reduces the time by a factor of 1000.

   Output:



```
backtrackingMRVfwd:
Execution Time: 0.0819433423087
Consistency Checks: 1554
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6,
```

   Above output in matrix format:

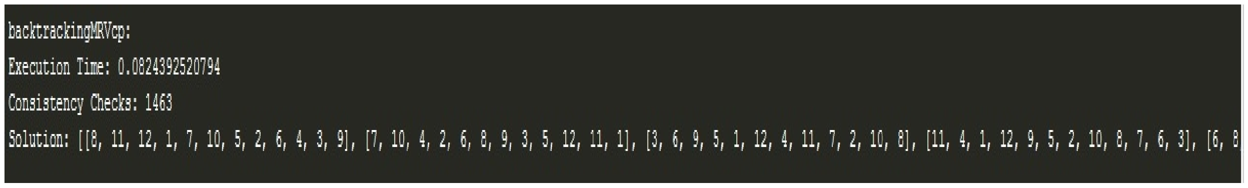


```
backtrackingMRVfwd:
Execution Time: 0.081848922402
Consistency Checks: 1554
Solution:
[[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9],
 [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1],
 [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8],
 [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3],
 [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11],
 [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12],
 [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7],
 [1, 9, 11, 3, 5, 7, 8, 6, 12, 10, 4, 2],
 [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5],
 [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4],
 [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6],
 [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]
```

4. **Backtracking + MRV + Constraint Propagation:**
   In Constraint Propagation, the AllDiff constraint is applied recursively till a valid value is selected for each variable. It differs from Forward checking in the way that forward checking applies arc consistency to all the constrained variables immediate to the input variable, but it stops there. However constraint propagation applies the constraint to other arcs affected by the previous step.

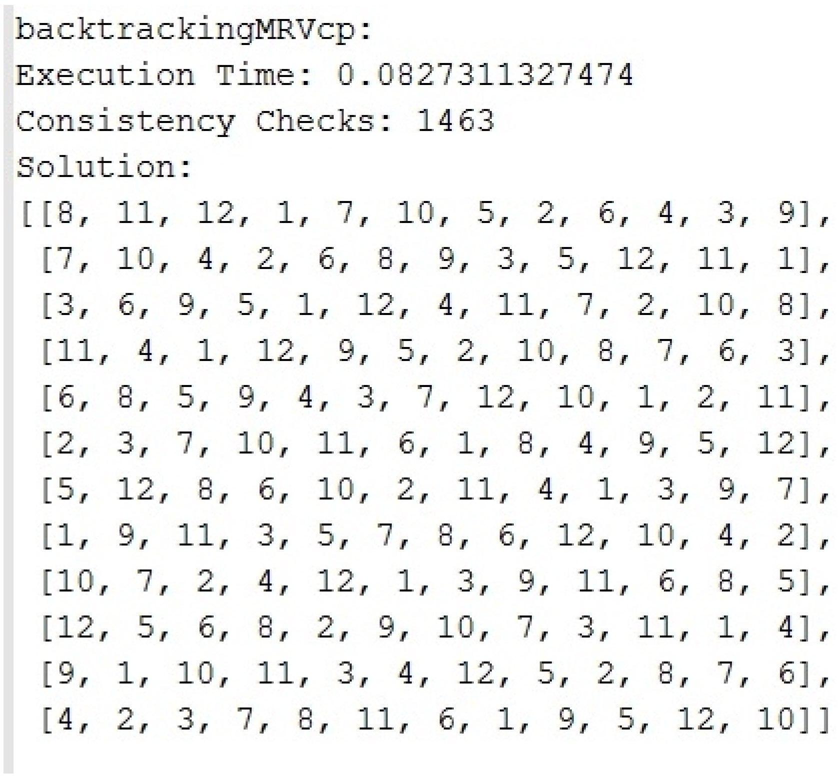
   For large input size, Constraint propagation will further prune down the search tree to avoid future failures and runs in small execution time compared to above discussed techniques.

   Output:



```
backtrackingMRVcp:
Execution Time: 0.0824392520794
Consistency Checks: 1463
Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 8
```

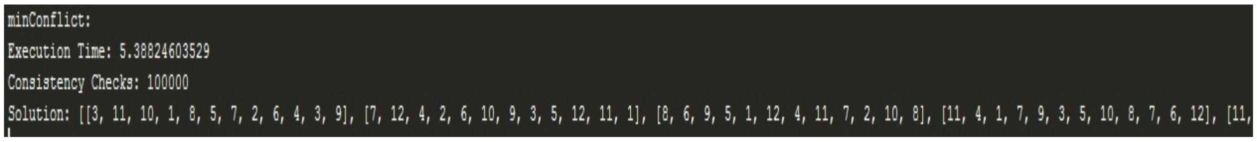
   Above output in matrix format:



```
backtrackingMRVcp:
Execution Time: 0.0827311327474
Consistency Checks: 1463
Solution:
[[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9],
 [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1],
 [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8],
 [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3],
 [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11],
 [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12],
 [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7],
 [1, 9, 11, 3, 5, 7, 8, 6, 12, 10, 4, 2],
 [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5],
 [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4],
 [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6],
 [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]
```

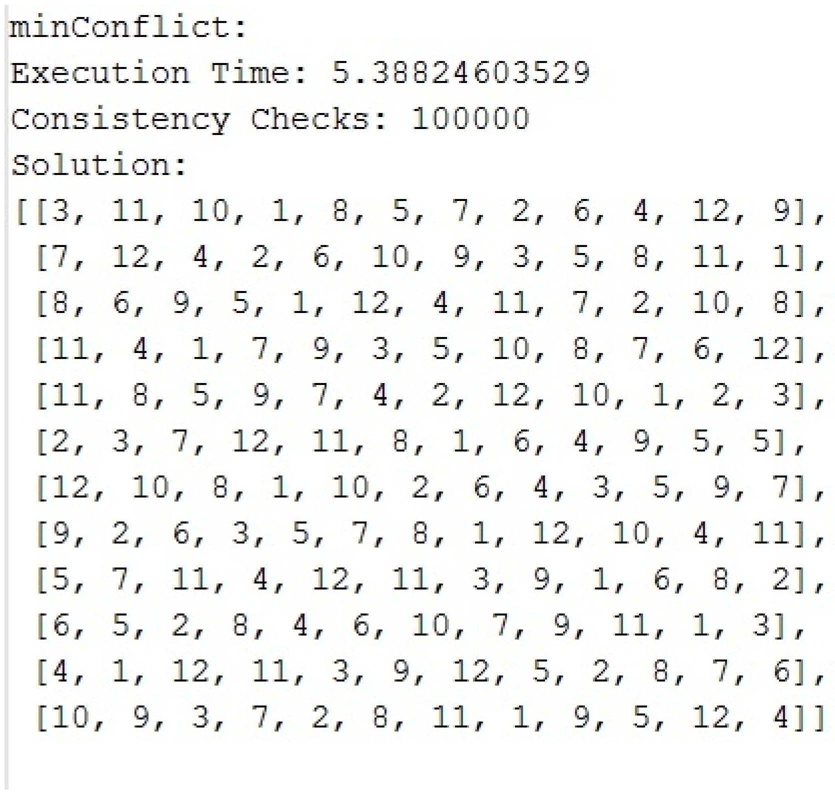
5. **Min-conflicts Heuristic:**
   In Min-conflict heuristic which has been implemented to solve the Sudoku game, for each position we check the number of conflicts with other variables in row, column and square. We select the variable with minimum conflicts.

   Output:



```
minConflict:
Execution Time: 5.38824603529
Consistency Checks: 100000
Solution: [[3, 11, 10, 1, 8, 5, 7, 2, 6, 4, 3, 9], [7, 12, 4, 2, 6, 10, 9, 3, 5, 12, 11, 1], [8, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 7, 9, 3, 5, 10, 8, 7, 6, 12], [11,
```

   Above output in matrix format:



```
minConflict:
Execution Time: 5.38824603529
Consistency Checks: 100000
Solution:
[[3, 11, 10, 1, 8, 5, 7, 2, 6, 4, 12, 9],
 [7, 12, 4, 2, 6, 10, 9, 3, 5, 8, 11, 1],
 [8, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8],
 [11, 4, 1, 7, 9, 3, 5, 10, 8, 7, 6, 12],
 [11, 8, 5, 9, 7, 4, 2, 12, 10, 1, 2, 3],
 [2, 3, 7, 12, 11, 8, 1, 6, 4, 9, 5, 5],
 [12, 10, 8, 1, 10, 2, 6, 4, 3, 5, 9, 7],
 [9, 2, 6, 3, 5, 7, 8, 1, 12, 10, 4, 11],
 [5, 7, 11, 4, 12, 11, 3, 9, 1, 6, 8, 2],
 [6, 5, 2, 8, 4, 6, 10, 7, 9, 11, 1, 3],
 [4, 1, 12, 11, 3, 9, 12, 5, 2, 8, 7, 6],
 [10, 9, 3, 7, 2, 8, 11, 1, 9, 5, 12, 4]]
```

Observation: So for large input size, Backtracking +MRV+ Constraint Propagation and Backtracking + MRV + Forward Checking technique gives the output in minimum time compared to other techniques. Constraint Propagation further prunes the search tree.