King Saud University College of Computer and Information Science Computer Science Department	
CSC361 Artificial Intelligence	2nd Semester 1445

Programming Project

Implementation Report

Section [44094]
Group [4]

Students	
Name	*ID*
Latifah Alessa	443200515
Lamees Alturki	443200862

I. Task Distribution

The table below shows task distribution among team members.

Student	***Tasks***
Latifah Alessa	Simple backtracking, Forward checking with MRV
Lamees Alturki	Generate grid, Forward checking

II. Overview

Tenner Grid puzzles pose an intriguing challenge in combinatorial problem solving, where the objective is to fill a grid with numbers while adhering to certain constraints.

Problem Formulation:

Variables: cells represented by it's coordinate on a matrix e.g {$C_{0,0}$, $C_{0,1}$, …}.

Domain: the set from 0-9

Constraints:

1. Numbers appear only once in a row.
2. Numbers may be repeated in columns.
3. Numbers in the columns must add up to the given sums.
4. Numbers in contiguous cells must be different.

This report presents the design and implementation of a program capable of generating and solving 3 by 10 sized Tenner Grid puzzles by modeling them as Constraint Satisfaction Problems.

III. Description

i. Backtracking

The recursive Step of this method is iterating over possible values from 0 to 9 and tries to assign them to the current cell if they satisfy the puzzle's rules. It then recursively calls itself to move to the next cell, repeat the process and if none of the values can be placed in the current cell without violating the rules, it backtracks by resetting the cell value to -1 and tries the next value in the previous recursive call. And so, on until it returns true if the algorithm has reached the end of the grid, indicating that the puzzle has been solved.

```
Function simpleBacktracking(grid, currentRow, currentColumn):
    Start timer

    If currentRow == LastRow AND currentColumn == LastColumn:
        Return true

    If currentColumn == LastColumn:
        currentColumn = FirstColumn
        currentRow = currentRow + 1

    If grid[currentRow][currentColumn] is not empty:
        Return simpleBacktracking(grid, currentRow, currentColumn + 1)

    For value from 1 to 9:
        If value satisfies rules for currentRow, currentColumn:
            Set grid[currentRow][currentColumn] = value
            If simpleBacktracking(grid, currentRow, currentColumn + 1):
                Stop timer
                Return true

        Unset grid[currentRow][currentColumn]

    Stop timer
    Return false
```

***Figure 1**: shows the pseudocode of the backtracking algorithm.*

ii. Forward checking

Firstly, the algorithm iterates through the grid to find the first empty cell, once an empty cell is found the algorithm iterates over the domain of possible values of said cell and checks if a number from the domain can be placed in the cell while satisfying the grid's rules and domain constraints. Otherwise, if no empty cell is found, it means the grid is already solved.
Before recursively calling the ForwardChecking function, the algorithm calls forwardCheckdomain to update the domains of neighboring cells based on the value just assigned to the current cell. Then if placing the value in the current cell is valid, the algorithm recursively calls itself to move to the next empty cell. If this recursive call returns true, then the puzzle has been solved. But if no valid value can be placed in the current cell, or if the recursive call returns false, indicating that no solution was found with the current assignment, the algorithm backtracks by resetting the current cell's value to -1. It also restores the domains of neighboring cells affected by the previous assignment.

```
Function ForwardChecking(grid, domains):
    Start timer

    FindEmptyCell(grid) // Find the first empty cell in the grid

    If grid is full:
        Stop timer
        Return true // Grid is solved

    For num from 0 to 9:
        If num is valid according to domain and rules:
            Assign num to the empty cell
            Update domain and constraints
            If ForwardChecking(grid, domains):
                Stop timer
                Return true // If the rest of the grid can be filled
            Backtrack // Undo the assignment and restore domain
    Return false // No valid number found for the empty cell
```

***Figure 2**: shows the pseudocode of the Forward checking algorithm.*

iii. Forward checking with MRV

This algorithm is almost identical to Forward Checking which we described above, but the main difference lies in the introduction of the concept of MRV, which refers to the unassigned cell with the fewest remaining values in its domain. The algorithm aims to prioritize such cells because they are likely to lead to quicker pruning of search branches.

```
Function ForwardCheckingMRV(grid, domains):
    Start timer

    FindUnassignedVariableWithMRV(grid, domains) // Find the unassigned variable with
                                                 //  the fewest remaining values
    If grid is full:
        Stop timer
        Return true // Grid is solved

    For num from 0 to 9:
        If num is valid according to domain and rules:
            Assign num to the empty cell
            Update domain and constraints
            If ForwardCheckingMRV(grid, domains):
                Stop timer
                Return true // If the rest of the grid can be filled
            Backtrack // Undo the assignment and restore domain
    Return false // No valid number found for the empty cell
```

***Figure 3**: shows the pseudocode of the Forward checking with MRV algorithm.*

IV. Sample Run

```
Welcome to the Tenner Grid generator and solver

 ---------Initial State---------
| 7  0  4 -1  2  9 -1  5  1  8 |
| 2 -1  5  0  6  4 -1 -1 -1 -1 |
| 0  8  4 -1 -1  1  9  5 -1  6 |
| 9  9 13  6 15 14 22 18  6 23 |

 ~~~~~~~~~Backtracking~~~~~~~~~

 ---------Final State---------
| 7  0  4  3  2  9  6  5  1  8 |
| 2  1  5  0  6  4  7  8  3  9 |
| 0  8  4  3  7  1  9  5  2  6 |
| 9  9 13  6 15 14 22 18  6 23 |
consistency: 587
assignments: 103
Time Used: 0.004 milliseconds

  ~~~~~~~~ForwardChecking~~~~~~~~

  ---------Final State---------
 | 7  0  4  3  2  9  6  5  1  8 |
 | 2  1  5  0  6  4  7  8  3  9 |
 | 0  8  4  3  7  1  9  5  2  6 |
 | 9  9 13  6 15 14 22 18  6 23 |
 consistency: 465
 assignments: 18
 Time Used: 0.0086 milliseconds

  ~~~~~~ForwardCheckingMRV~~~~~~

  ---------Final State---------
 | 7  0  4  3  2  9  6  5  1  8 |
 | 2  1  5  0  6  4  7  8  3  9 |
 | 0  8  4  3  7  1  9  5  2  6 |
 | 9  9 13  6 15 14 22 18  6 23 |
 consistency: 632
 assignments: 24
 Time Used: 0.0128 milliseconds
```

V. Analyzing The Algorithms

iv. Comparison

```
---------Initial State---------
| 3  0  1  2  9  8  4  7  5  6 |
| 6  4  3 -1  5  1  2  9  8  0 |
| 1 -1  8  2  4 -1  5  0 -1  9 |
|10 11 12 11 18 15 11 16 16 15 |

~~~~~~~~~Backtracking~~~~~~~~~

---------Final State---------
| 3  0  1  2  9  8  4  7  5  6 |
| 6  4  3  7  5  1  2  9  8  0 |
| 1  7  8  2  4  6  5  0  3  9 |
|10 11 12 11 18 15 11 16 16 15 |
consistency: 174
assignments: 27
Time Used: 3.0E-4 milliseconds

~~~~~~~~ForwardChecking~~~~~~~~

---------Final State---------
| 3  0  1  2  9  8  4  7  5  6 |
| 6  4  3  7  5  1  2  9  8  0 |
| 1  7  8  2  4  6  5  0  3  9 |
|10 11 12 11 18 15 11 16 16 15 |
consistency: 174
assignments: 4
Time Used: 9.0E-4 milliseconds

~~~~~~ForwardCheckingMRV~~~~~~

---------Final State---------
| 3  0  1  2  9  8  4  7  5  6 |
| 6  4  3  7  5  1  2  9  8  0 |
| 1  7  8  2  4  6  5  0  3  9 |
|10 11 12 11 18 15 11 16 16 15 |
consistency: 174
assignments: 4
Time Used: 0.001 milliseconds
```

```
---------Initial State---------
|-1  5 -1 -1  3 -1 -1  6 -1  8 |
|-1 -1  7  0  6  1 -1  3 -1  5 |
| 0 -1 -1  1  4  2 -1  6  8  3 |
| 2 14 25  2 13  5 22 15 21 16 |

~~~~~~~~~Backtracking~~~~~~~~~

---------Final State---------
| 0  5  9  1  3  2  7  6  4  8 |
| 2  4  7  0  6  1  8  3  9  5 |
| 0  5  9  1  4  2  7  6  8  3 |
| 2 14 25  2 13  5 22 15 21 16 |
consistency: 2777
assignments: 465
Time Used: 8.0E-4 milliseconds

~~~~~~~~ForwardChecking~~~~~~~~

---------Final State---------
| 0  5  9  1  3  2  7  6  4  8 |
| 2  4  7  0  6  1  8  3  9  5 |
| 0  5  9  1  4  2  7  6  8  3 |
| 2 14 25  2 13  5 22 15 21 16 |
consistency: 2338
assignments: 83
Time Used: 0.0012 milliseconds

~~~~~~ForwardCheckingMRV~~~~~~

---------Final State---------
| 0  5  9  1  3  2  7  6  4  8 |
| 2  4  7  0  6  1  8  3  9  5 |
| 0  5  9  1  4  2  7  6  8  3 |
| 2 14 25  2 13  5 22 15 21 16 |
consistency: 2463
assignments: 85
Time Used: 0.0013 milliseconds
```

```
 ---------Initial State---------
| 8  7  5  9  1  6  3  0  2  4 |
|-1  3 -1 -1 -1 -1 -1 -1 -1 -1 |
|-1 -1 -1 -1  0 -1  4 -1  6 -1 |
|16 17 15 21  3 18 15  6 15  9 |

 ~~~~~~~~~Backtracking~~~~~~~~~

 ---------Final State---------
| 8  7  5  9  1  6  3  0  2  4 |
| 6  3  1  4  2  9  8  5  7  0 |
| 2  7  9  8  0  3  4  1  6  5 |
|16 17 15 21  3 18 15  6 15  9 |
consistency: 69419
assignments: 11863
Time Used: 9.0E-4 milliseconds

 ~~~~~~~~ForwardChecking~~~~~~~~

 ---------Final State---------
| 8  7  5  9  1  6  3  0  2  4 |
| 6  3  1  4  2  9  8  5  7  0 |
| 2  7  9  8  0  3  4  1  6  5 |
|16 17 15 21  3 18 15  6 15  9 |
consistency: 42820
assignments: 2156
Time Used: 0.0016 milliseconds

 ~~~~~~ForwardCheckingMRV~~~~~~

 ---------Final State---------
| 8  7  5  9  1  6  3  0  2  4 |
| 6  3  1  4  2  9  8  5  7  0 |
| 2  7  9  8  0  3  4  1  6  5 |
|16 17 15 21  3 18 15  6 15  9 |
consistency: 42430
assignments: 2160
Time Used: 0.0015 milliseconds
```

```
 ---------Initial State---------
|-1  9 -1 -1 -1 -1 -1 -1 -1 -1 |
|-1 -1 -1  1 -1 -1  5  6  3  0 |
| 9 -1 -1 -1 -1 -1  2  8 -1  5 |
|19 17 19  6 14 18  7 21  8  6 |

 ~~~~~~~~~Backtracking~~~~~~~~~

 ---------Final State---------
| 3  9  8  2  5  6  0  7  4  1 |
| 7  2  4  1  9  8  5  6  3  0 |
| 9  6  7  3  0  4  2  8  1  5 |
|19 17 19  6 14 18  7 21  8  6 |
consistency: 589769
assignments: 97941
Time Used: 0.0051 milliseconds

 ~~~~~~~~ForwardChecking~~~~~~~~

 ---------Final State---------
| 3  9  8  2  5  6  0  7  4  1 |
| 7  2  4  1  9  8  5  6  3  0 |
| 9  6  7  3  0  4  2  8  1  5 |
|19 17 19  6 14 18  7 21  8  6 |
consistency: 383379
assignments: 17808
Time Used: 0.0025 milliseconds

 ~~~~~~ForwardCheckingMRV~~~~~~

 ---------Final State---------
| 3  9  8  2  5  6  0  7  4  1 |
| 7  2  4  1  9  8  5  6  3  0 |
| 9  6  7  3  0  4  2  8  1  5 |
|19 17 19  6 14 18  7 21  8  6 |
consistency: 361626
assignments: 15690
Time Used: 0.0038 milliseconds
```

```
 ---------Initial State---------
| 5 -1  9 -1  6  4 -1  0 -1 -1 |
|-1 -1  1 -1  9  8  2 -1 -1  4 |
| 5  7  4 -1 -1  0 -1 -1  9  2 |
|16 14 14 13 16 12  8 15 20  7 |

 ~~~~~~~~~Backtracking~~~~~~~~~

 ---------Final State---------
| 5  7  9  2  6  4  3  0  8  1 |
| 6  0  1  5  9  8  2  7  3  4 |
| 5  7  4  6  1  0  3  8  9  2 |
|16 14 14 13 16 12  8 15 20  7 |
consistency: 6138
assignments: 1075
Time Used: 7.0E-4 milliseconds

 ~~~~~~~~ForwardChecking~~~~~~~~

 ---------Final State---------
| 5  7  9  2  6  4  3  0  8  1 |
| 6  0  1  5  9  8  2  7  3  4 |
| 5  7  4  6  1  0  3  8  9  2 |
|16 14 14 13 16 12  8 15 20  7 |
consistency: 4653
assignments: 196
Time Used: 0.002 milliseconds

 ~~~~~~ForwardCheckingMRV~~~~~~

 ---------Final State---------
| 5  7  9  2  6  4  3  0  8  1 |
| 6  0  1  5  9  8  2  7  3  4 |
| 5  7  4  6  1  0  3  8  9  2 |
|16 14 14 13 16 12  8 15 20  7 |
consistency: 4232
assignments: 168
Time Used: 0.0019 milliseconds
```

	No. consistency checks					**Median**
	Game #1	Game #2	Game #3	Game #4	Game #5	
Backtracking	174	2777	6138	589769	69419	6138
Forward checking	174	2338	4653	383379	42820	4653
FC + MRV	174	2463	4232	361626	42430	4232

v. Analysis

It is observed that Simple Backtracking still solves the Tenner Grid problem despite being the least efficient among the three tested algorithms. Since, it requires the highest median number of consistency checks, indicating that it explores a larger portion of the search space before finding a solution. This suggests that Simple Backtracking may struggle with larger or more complex instances of the Tenner Grid problem due to its exhaustive search approach. Forward Checking improves upon Simple Backtracking by pruning the search space more aggressively. As a result, fewer consistency checks are required to find a solution to the Tenner Grid problem. However, it was still outperformed by Forward Checking with MRV, suggesting that there is still room for improvement in reducing the search space further.

Forward Checking with MRV stands out as the most efficient algorithm for solving the Tenner Grid problem in terms of the median number of consistency checks required. By prioritizing variables with the fewest remaining values in their domains, this algorithm effectively reduces the search space, leading to quicker convergence to a solution. This makes it particularly well suited for larger or more complex instances of the Tenner Grid problem, where efficiency is crucial.

To conclude, while all three algorithms can solve the Tenner Grid problem, Forward Checking with MRV emerges as the most efficient option, followed by Forward Checking and then Simple Backtracking. Depending on the size and complexity of the problem instance, choosing the appropriate algorithm can significantly impact the efficiency and scalability of the solution.

VI. Student Peer Evaluation

The table below illustrate Student-Peer-Evaluation across two different aspects.
(1: fully satisfied, 0.5: partially satisfied, 0: not satisfied)

Name of evaluator		Evaluation for each student	
		Level of commitment	
		Latifah	**Lamees**
	Latifah		1
	Lamees	1	
		Professional behavior towards team	
		Latifah	**Lamees**
	Latifah		1
	Lamees	1	