**Assignment 2: Implement Constraint Satisfaction Problem (CSP)**

**Problem Statement:**

Implement a solution to a constraint satisfaction problem (CSP) using backtracking search and constraint propagation techniques, focusing on solving problems like sudoku, map coloring, or scheduling.

**Objectives:**

* To solve a CSP using backtracking search with constraint propagation.
* To implement an efficient state representation and variable selection strategy.
* To optimize the solution by minimizing search space through constraint propagation.
* To compare the performance of different heuristics for variable and value selection.

**Theory:**

**What is a CSP?**

A Constraint Satisfaction Problem (CSP) involves finding values for variables that satisfy a set of constraints. In a CSP, the problem is structured as follows:

* **Variables**: Elements that need to be assigned values (e.g., cells in Sudoku).
* **Domains**: The set of possible values each variable can take (e.g., numbers 1-9 for Sudoku).
* **Constraints**: Rules that dictate how variables can be assigned values (e.g., no two cells in the same row, column, or sub-grid can have the same value in Sudoku).

**Methodology**

1. **Define Variables, Domains, and Constraints**:

* Variables represent each cell in the Sudoku grid.
* The domain for each variable is the numbers 1 to 9.
* Constraints ensure that no two cells in the same row, column, or 3x3 sub-grid contain the same number.

1. **Start with an Empty Assignment**:
   * Initialize an empty Sudoku grid where variables (cells) will be assigned values as you progress.
2. **Use Backtracking**:
   * Assign a value to a variable, then check if it satisfies all constraints.
   * If valid, move to the next variable; otherwise, backtrack and try a different value.
   * Continue this process until a valid solution is found or all options are exhausted.
3. **Continue Until a Valid Solution is Found**:
   * If a valid solution is reached (all constraints satisfied), output the result.
   * If no valid solution is possible, report the failure.

**Working Principle / Algorithm**

Here’s a basic outline of the backtracking algorithm to solve Sudoku:

1. **Find the Next Empty Cell**:
   * Scan the grid for the next cell that needs a value.
2. **Try Each Possible Value**:
   * For each number from 1 to 9:
     + Assign the number to the empty cell.
     + Check if the assignment is valid (i.e., it does not violate any Sudoku constraints).
     + If valid, recursively call the function to assign values to the next cell.
3. **Backtrack If Necessary**:
   * If the recursive call leads to a solution, propagate that solution upwards.
   * If not, reset the cell and try the next number.
4. **End Condition**:
   * If all cells are filled correctly, the puzzle is solved.

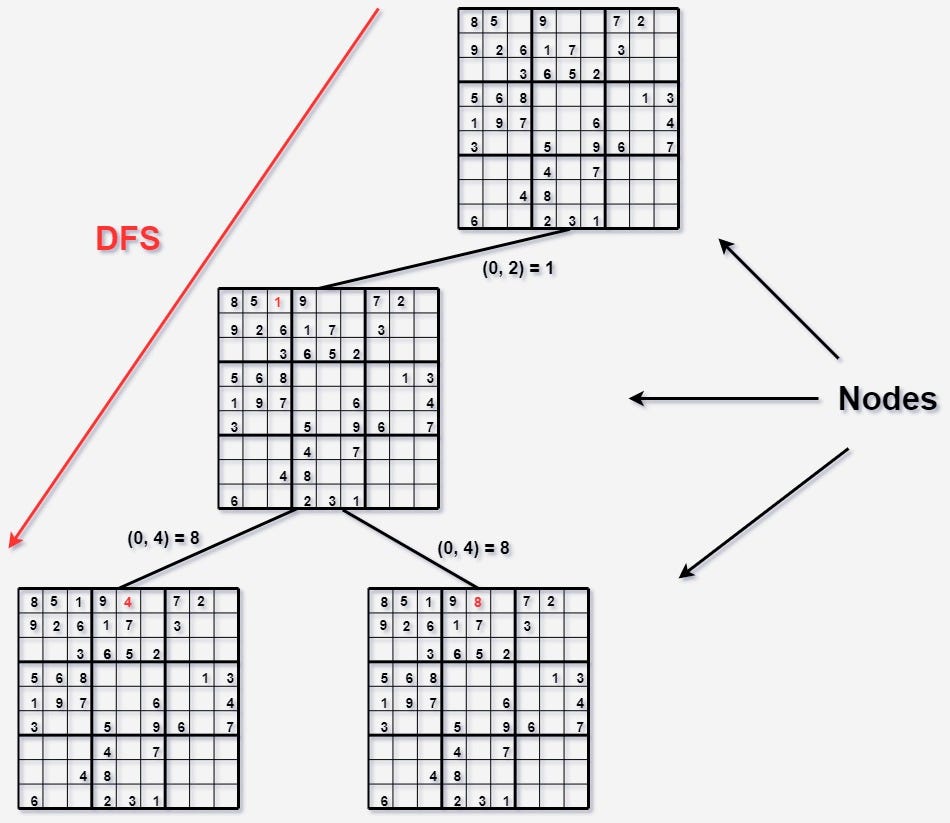
**Advantages**

* Structured Approach: CSPs provide a clear and systematic framework for problem-solving, using well-defined variables, domains, and constraints.
* Scalable for Multiple Variables: Backtracking, enhanced with heuristics, is highly effective for problems with numerous variables and constraints.
* Flexibility: CSPs can easily adapt to various problem types, such as Sudoku, map coloring, and scheduling.

**Disadvantages / Limitations**

* **Performance**: Backtracking can be computationally expensive and slow for large CSPs, particularly as the number of variables and constraints increases.
* **Exponential Time Complexity**: The worst-case time complexity can grow exponentially with the size of the problem.

**Diagram**



**Conclusion**

CSPs offer a systematic way to approach complex problems by framing them in terms of constraints. The backtracking technique allows us to explore possible assignments efficiently, making it a valuable tool for solving problems like Sudoku and many others.