Experiment No.-06

Objective: Solve Constraint Satisfaction Problems.

Theory: A Constraint Satisfaction Problem (CSP) is a mathematical problem defined as a set of objects whose state must satisfy a number of constraints or limitations.
It is widely used in Artificial Intelligence, operations research, and optimization.
A CSP is represented by:
• X = {X1, X2, ..., Xn} → a set of variables
• D = {D1, D2, ..., Dn} → domains of possible values for each variable
• C = {C1, C2, ..., Ck} → a set of constraints specifying allowable combinations of values
A solution to a CSP is an assignment of values to all variables that satisfies all constraints.

➢ Types of CSPs:
• Discrete CSPs: Variables take values from a finite domain (e.g., Sudoku, N-Queens).
• Continuous CSPs: Variables have continuous domains (e.g., scheduling problems).
• Dynamic CSPs: Constraints or variables may change over time.

➢ Common Techniques for Solving CSPs:
• Backtracking Search: Systematically explores possible variable assignments and backtracks when constraints are violated.
• Forward Checking: Prevents future conflicts by checking consistency during assignment.
• Arc Consistency (AC-3 Algorithm): Removes inconsistent values from domains.
• Heuristics:
o Minimum Remaining Values (MRV) — choose the variable with the fewest remaining legal values.
o Least Constraining Value (LCV) — choose the value that rules out the fewest options for neighboring variables.

➢ Applications:
• Map coloring
• Sudoku solving
• Scheduling and timetabling
• Resource allocation
• N-Queens problem

Algorithm: Steps:
Step 1: Initialization
• Define the set of variables and their possible domains.
• Define the constraints between variables.
Step 2: Selection
• Choose an unassigned variable.

Step 3: Assignment
• Assign a value from its domain.
Step 4: Constraint Checking
• Check if the assignment is consistent with the constraints.
Step 5: Backtrack
• If a constraint is violated, undo the last assignment and try a different value.
Step 6: Termination
• If all variables are assigned valid values, output the solution.
• If no valid assignment exists, report failure.
Time Complexity:
Worst case — O(dn) where d = size of domain, n = number of variables.
Space Complexity:
O(n) for recursion stack and variable assignments.

Program:

```
def is_safe(assignment, var, value, constraints):
for neighbor in constraints[var]:
if neighbor in assignment and assignment[neighbor] == value:
return False
return True

def backtrack(variables, domains, constraints, assignment):
if len(assignment) == len(variables):
return assignment # All variables assigned

var = [v for v in variables if v not in assignment][0]

for value in domains[var]:
if is_safe(assignment, var, value, constraints):
assignment[var] = value
result = backtrack(variables, domains, constraints, assignment)
if result:
return result
del assignment[var] # backtrack
return None

def main():
variables = ['WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T']

domains = {
'WA': ['Red', 'Green', 'Blue'],
'NT': ['Red', 'Green', 'Blue'],
'SA': ['Red', 'Green', 'Blue'],
'Q': ['Red', 'Green', 'Blue'],
```

```
'NSW': ['Red', 'Green', 'Blue'],
'V': ['Red', 'Green', 'Blue'],
'T': ['Red', 'Green', 'Blue']
}

constraints = {
'WA': ['NT', 'SA'],
'NT': ['WA', 'SA', 'Q'],
'SA': ['WA', 'NT', 'Q', 'NSW', 'V'],
'Q': ['NT', 'SA', 'NSW'],
'NSW': ['Q', 'SA', 'V'],
'V': ['SA', 'NSW'],
'T': []
}

assignment = {}
solution = backtrack(variables, domains, constraints, assignment)

if solution:
print("Solution Found:")
for state, color in solution.items():
print(f"{state} → {color}")
else:
print("No solution exists.")

if __name__ == "__main__":
main()
```

Output:

```
Solution Found:
WA → Red
NT → Green
SA → Blue
Q → Red
NSW → Green
V → Red
T → Red
```