Artificial Intelligence

BS (CS) _SP_2024

Lab_08 Manual

Learning Objectives:

1. CSP (Constraint Satisfaction Problem)

Introduction to CSP

Constraint Satisfaction Problems (CSPs) involve finding values for variables while satisfying a set of **constraints**. CSPs are widely used in scheduling, map coloring, sudoku solving, and many other applications.

Examples of CSPs

- **Map Coloring**: Assign colors to regions such that adjacent regions have different colors.
- **Sudoku**: Fill a grid with numbers so that each row, column, and sub-grid contain unique numbers.
- **N-Queens Problem**: Place N queens on an NxN chessboard so that no two attack each other.

Understanding CSPs

A Constraint Satisfaction Problem (CSP) consists of:

- A set of **variables** $X = \{X_1, X_2, X_3 \dots X_n\}$
- A set of **domains** $D = \{D_1, D_2, D_3 \dots D_n\}$, where each D_i contains possible values for X_i.
- A set of **constraints** C_i that restrict variable assignments. Each constraint $C_i$consists of a pair <scope, rel>, where scope is a tuple of variables and rel is the relation, either represented explicitly or abstractly

Example: Map coloring problem, where:

- **Variables**: Different regions on a map.
- **Domains**: Possible colors (e.g., Red, Blue, Green).
- **Constraints**: Adjacent regions cannot have the same color.

Each state in a CSP is defined by an assignment of values to some or all of the variables.

- An assignment that does not violate any constraints is called a **consistent** or **legal** assignment
- A **complete assignment** is one in which every variable is assigned
- A solution to a CSP is **consistent** and **complete assignment**
- Allows useful general-purpose algorithms with more power than standard search algorithms

Techniques to Solve CSPs

1. Backtracking Algorithm

A brute-force approach that assigns values to variables one by one, backtracking if a constraint is violated.

2. Heuristics for CSPs

To improve efficiency, CSP solvers use heuristics:

- **Minimum Remaining Values (MRV)**: Choose the variable with the fewest legal values.
- **Most Constraining Variable**: If a tie exists in MRV, choose the variable that restricts others the most.
- **Least Constraining Value (LCV)**: When selecting a value, choose the one that leaves the most options for remaining variables.

3. Consistency Techniques

- **Node Consistency**: Ensures that all values in a variable's domain satisfy unary constraints.
- **Arc Consistency (AC-3 Algorithm)**: Ensures that for every value of a variable, there exists a valid value in the connected variable.

Pseudocodes

Minimum Remaining Values (MRV) Heuristic

The MRV heuristic selects the variable with the fewest legal values first (also called the fail-first heuristic).

Pseudocode for MRV:

```
function MRV(variables, domains, constraints):
    min_var = None
    min_values = infinity
    for var in variables:
        if var is not assigned:
            num_values = count_valid_values(var, domains, constraints)
            if num_values < min_values:
                min_values = num_values
                min_var = var
    return min_var
```

Most Constraining Variable Heuristic

If there is a tie in MRV, the **Most Constraining Variable** heuristic chooses the variable that imposes the most restrictions on other unassigned variables.

Pseudocode for Most Constraining Variable:

```
function MostConstrainingVariable(variables, constraints):
    max_var = None
    max_constraints = -1
    for var in variables:
        if var is not assigned:
            num_constraints = count_constraints_on_other_vars(var, constraints)
            if num_constraints > max_constraints:
                max_constraints = num_constraints
                max_var = var
    return max_var
```

Least Constraining Value Heuristic

This heuristic chooses the value that **rules out the fewest values** in remaining variables.

Pseudocode for Least Constraining Value:

```
function LeastConstrainingValue(var, domains, constraints):
    values = get_domain(var, domains)
    sorted_values = sort_by_fewest_constraints(values, var, constraints)
    return sorted_values
```

Node Consistency

A variable is node-consistent if all values in its domain satisfy unary constraints.

Pseudocode for Node Consistency:

```
function NodeConsistency(variables, domains, constraints):
    for var in variables:
        for value in domains[var]:
            if not satisfies_unary_constraints(var, value, constraints):
                remove(value, domains[var])
```

Arc Consistency (AC-3 Algorithm)

A variable is **arc-consistent** with if for every value in 's domain, there exists a valid value in 's domain.

Pseudocode for AC-3 Algorithm:

```
function AC3(csp):
    queue = all_arcs(csp)
    while queue is not empty:
        (X, Y) = queue.pop()
        if RemoveInconsistentValues(X, Y, csp):
            for each neighbor Z of X:
                queue.add((Z, X))

def RemoveInconsistentValues(X, Y, csp):
    removed = False
    for x in domain[X]:
        if no y in domain[Y] satisfies constraint(X, Y, x, y):
            domain[X].remove(x)
            removed = True
    return removed
```

Backtracking Algorithm for CSP

A **backtracking algorithm** is a depth-first search that assigns values to variables while ensuring constraints are satisfied.

Pseudocode for Backtracking:

```
function BacktrackingSearch(assignment, variables, domains, constraints):
    if assignment is complete:
        return assignment
    var = select_unassigned_variable(variables, assignment)
    for value in order_domain_values(var, assignment, domains):
        if is_consistent(var, value, assignment, constraints):
            assignment[var] = value
            result = BacktrackingSearch(assignment, variables, domains,
            constraints)
            if result is not failure:
                return result
            assignment.remove(var)
    return failure
```

Summary:

- **MRV** selects the most constrained variable.
- **Most Constraining Variable** helps break ties.
- **Least Constraining Value** picks values that minimize conflicts.
- **Node Consistency** ensures unary constraints are met.
- **Arc Consistency (AC-3)** reduces domain values to maintain constraints.
- **Backtracking Search** is used to find solutions efficiently.