**ASSIGNMENT HELP**

**MANUAL**



SUBMITTED

TO

VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE

FOR THE SKILL AND COMPETENCY EVALUATION OF

ARTIFICIAL INTELLIGENCE [CAUA31201]

IN

**CSE AI DEPARTMENT**

BY

**Vedant Rakesh Mukhekar**

**Class: T.Y. BTech Division: A Batch: A2**

**Batch Teacher**

**Dr. ANURADHA YENKIKAR.**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **CONTENTS** | **PAGE NO.** |
| **1** | **PROBLEM STATEMENT** | **4-5** |
| **2** | **LIBRARY USED** | **5-6** |
| **3** | **THEORY** | **6-7** |
| **4** | **METHODOLOGY** | **7-14** |
| **5** | **ADVANTAGES & DISADVANTAGES** | **14-15** |
| **6** | **WORKING** | **16-17** |
| **7** | **DIAGRAM** | **17-18** |
| **8** | **CONCLUSION** | **18-19** |

### ****Problem Statement****

A **Constraint Satisfaction Problem (CSP)** consists of a set of variables, each with a domain of possible values, and a set of constraints specifying allowable combinations of values. The goal is to find an assignment of values to variables such that all constraints are satisfied.

In this experiment, we will implement a CSP solver using **Backtracking Search** and will optionally improve it with heuristics like **MRV (Minimum Remaining Values)** and **AC-3 (Arc Consistency)**.

### ****Libraries Used****

* **Python Standard Libraries**:
  + collections: Used for managing queues in constraint propagation (AC-3).
  + copy: Used to handle deep copying of assignments or domain states.

**Optional**:

* + Visualization or deeper analysis can use matplotlib or numpy.

### ****Theory****

A **CSP** involves:

1. **Variables**: A set of unknowns to be assigned values.
2. **Domains**: Each variable has a domain of values it can be assigned from.
3. **Constraints**: A set of rules that restrict combinations of variable values.

Two primary algorithms are used:

* **Backtracking Search**: This explores possible variable assignments, backtracking when constraints are violated.
* **AC-3 (Arc Consistency)**: This reduces the domain of variables by removing values that are inconsistent with constraints.

#### ****Example CSP Problem: Map Coloring****

In a map-coloring problem, each country must be assigned a color such that no two neighboring countries have the same color. The variables represent the countries, the domains are the available colors, and the constraints enforce that neighboring countries cannot have the same color.

### ****Methodology****

1. **Define Variables**: Start by defining variables that represent the unknowns (e.g., countries in map coloring).
2. **Define Domains**: Each variable has a domain of values (e.g., a list of colors in map coloring).
3. **Define Constraints**: Specify constraints that limit which values can be assigned together (e.g., neighboring countries cannot share the same color).
4. **Backtracking Search Algorithm**:
   * **Step 1**: Select an unassigned variable.
   * **Step 2**: Assign a value to the variable from its domain.
   * **Step 3**: Check if the assignment satisfies the constraints.
   * **Step 4**: If consistent, continue assigning values to other variables; if not, backtrack.
5. **Heuristics (Optional)**:
   * **MRV (Minimum Remaining Values)**: Select the variable with the fewest possible remaining values.
   * **LCV (Least Constraining Value)**: Choose the value that rules out the fewest choices for neighboring variables.
6. **Constraint Propagation (AC-3)**: After assigning values, use constraint propagation to reduce domains by eliminating inconsistent values.

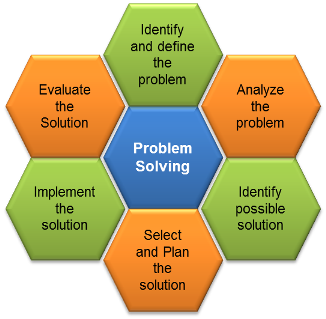
### ****Advantages & Disadvantages****

* **Advantages**:
  + **Backtracking Search** is easy to implement and provides a complete solution.
  + **Heuristics** like MRV and LCV can significantly reduce the search space.
  + **AC-3** improves efficiency by pruning inconsistent values early.
* **Disadvantages**:
  + **Backtracking Search** can be inefficient without heuristics, potentially exploring many redundant states.
  + **AC-3** adds overhead, which may not always be beneficial for simple problems.

### ****Working****

1. Start by defining the variables and domains.
2. Implement a backtracking search to explore possible assignments.
3. Use heuristics to optimize variable and value selection (MRV, LCV).
4. Optionally implement AC-3 for constraint propagation.

### ****Diagram****



### ****Code Example (Backtracking Search with Optional MRV and AC-3)****

python

Copy code

class CSP:

def \_\_init\_\_(self, variables, domains, constraints):

self.variables = variables # List of variables

self.domains = {var: list(domains) for var in variables} # Domains for each variable

self.constraints = constraints # Constraints (function that checks the validity of an assignment)

self.assignment = {} # Stores the current assignment of values

def is\_consistent(self, var, value):

"""Check if assigning value to var is consistent with the constraints."""

for (v1, v2), constraint\_func in self.constraints.items():

if v1 == var or v2 == var:

if v1 in self.assignment and v2 in self.assignment:

if not constraint\_func(self.assignment[v1], self.assignment[v2]):

return False

return True

def select\_unassigned\_variable(self):

"""Select a variable that hasn't been assigned a value yet (simple heuristic: first unassigned variable)."""

for var in self.variables:

if var not in self.assignment:

return var

return None

def backtrack(self):

"""Backtracking search algorithm."""

if len(self.assignment) == len(self.variables):

return self.assignment # All variables are assigned

var = self.select\_unassigned\_variable()

for value in self.domains[var]:

if self.is\_consistent(var, value):

self.assignment[var] = value # Assign value

result = self.backtrack() # Recurse

if result:

return result

del self.assignment[var] # Backtrack

return None # No valid assignment found

# Example: Map coloring problem

variables = ['A', 'B', 'C']

domains = ['Red', 'Green', 'Blue']

constraints = {

('A', 'B'): lambda a, b: a != b,

('B', 'C'): lambda b, c: b != c,

('A', 'C'): lambda a, c: a != c

}

csp = CSP(variables, domains, constraints)

solution = csp.backtrack()

print("Solution:", solution)

### ****Conclusion****

Implementing a **Constraint Satisfaction Problem (CSP)** using **Backtracking Search** is effective for small to medium-sized problems. Heuristics like **MRV** and **LCV** can significantly enhance performance by reducing the search space. Additionally, using **AC-3** helps enforce consistency, ensuring that no unnecessary states are explored.