Q4: k-CSPs

Q4: k-CSPs

Let a k-CSP be a CSP where the solution is allowed to have k violated constraints. We would like to modify the classic CSP algorithm to solve k-CSPs. The classic backtracking algorithm is shown below. To modify it to solve k-CSPs, we need to change line 15. Note that k is used to denote the number of allowable violated constraints.

```
1: function K-CSP-Backtracking(csp, k)
2:     return Recursive-Backtracking({}, csp, k)
3: end function
```

```
1: function Recursive-Backtracking(assignment, csp, k)
2:     if assignment is complete then
3:         return assignment
4:     end if
5:     var ← Select-Unassigned-Variable(Variables[csp], assignment, csp)
6:     for each value in Order-Domain-Values(var, assignment, csp) do
7:         if value is consistent with assignment given Constraints(csp) then
8:             add {var = value} to assignment
9:             result ← Recursive-Backtracking(assignment, csp, k)
10:            if result ≠ failure then
11:                return result
12:            end if
13:            remove {var = value} from assignment
14:        else
15:            continue
16:        end if
17:    end for
18:    return failure
19: end function
```

Part 1

0.0/5.0 points (graded)

If each of the following blocks of code were to replace line 15, which code block(s) would yield a correct algorithm for solving k-CSPS?

- [x] ✔

```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    result ← Recursive-Backtracking(
                        assignment, csp, k)
    if result ≠ failure then
        return result
    end if
end if
remove {var = value} from assignment
```

- [ ]

```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    Filter-Domains-with-Forward-Checking()
    result ← Recursive-Backtracking(
                        assignment, csp, k)
    if result ≠ failure then
        return result
    end if
    Undo-Filter-Domains-with-Fwd-Checking()
end if
remove {var = value} from assignment
```

```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    if Is-Tree(Unassigned-Variables[csp]) then
        result ← Tree-Structured-CSP-Algorithm(
                            assignment, csp)
    else
        result ← Recursive-Backtracking(
                            assignment, csp, k)
    end if
    if result ≠ failure then
        return result
    end if
end if
remove {var = value} from assignment
```

```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    Filter-Domains-with-Arc-Consistency()
    result ← Recursive-Backtracking(
                        assignment, csp, k)
    if result ≠ failure then
        return result
    end if
    Undo-Filter-Domains-with-Arc-Consistency()
end if
remove {var = value} from assignment
```

- None of the code blocks

Explanation

For this formulation, we want to only backtrack when more than k constraints are violated. If there are fewer than k, we can assign the current variable to a constrained value and continue the algorithm. The first code block provides the correct implementation. The second option using a tree structured CSP algorithm will incorrectly return failure if there is no solution with only i + 1 constraints violated (where i + 1 may be less than k). The third and fourth option filter the domains of the remaining variables. If a variable's domain is filtered, then the recursive-backtracking call will only consider values in the filtered domain rather than values which violate a constraint, potentially causing the code to miss the solution.

Submit You have used 0 of 2 attempts

Answers are displayed within the problem