**🎓 Assignment: Understanding Constraint Satisfaction Problems (CSPs)**

**📌 Section A: Conceptual Understanding**

**Instructions**: Answer the following in your own words.

1. **What is a Constraint Satisfaction Problem (CSP)?** Give two real-life examples that can be modeled as CSPs.

**Variables** – A set of unknowns that need to be assigned values.

**Domains** – A set of possible values each variable can take.

**Constraints** – Rules that restrict the combinations of values the variables can take.

**Real Life Example:**

1.Exam Scheduling

2.Sudoku Puzzle

1. **Explain the three main components of a CSP.**  
   Include a brief explanation of each with your own example (not covered in class).

**1. Variables**

* These are the unknowns we need to assign values .

**2. Domains**

* The domain of a variable is the set of possible values it can take.

**3. Constraints**

* These are the rules that restrict which combinations of variable assignments are allowed.

1. **Differentiate between:**
   * Unary, Binary, and Higher-order constraints
   * Hard constraints vs Soft constraints

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Unary** | One variable | X≠3X \neq 3X=3 |
| **Binary** | Two variables | A≠BA \neq BA=B |
| **Higher-Order** | Three or more variables | AllDifferent(X, Y, Z) |
| **Hard Constraint** | Must be satisfied | No overlapping exams |
| **Soft Constraint** | Preferable to satisfy, but optional | Prefer Alice near a window |

1. **What is a constraint graph?** Draw a sample constraint graph using the following constraint set:

A ≠ B, B ≠ C, A ≠ C, C ≠ D

A **constraint graph** is a visual representation of a Constraint Satisfaction Problem (CSP)

constraint graph using the following constraint set:

A ≠ B, B ≠ C, A ≠ C, C ≠ D

A

/ \

B---C

\

D

**📌 Section B: Short Answer and Reasoning**

1. You are given:
   * Variables: X, Y
   * Domains: D(X) = {1, 2, 3}, D(Y) = {2, 3, 4}
   * Constraint: X < Y
2. Is the assignment X=3, Y=2 valid? Why or why not?

No, the assignment X = 3, Y = 2 is ***not valid***.  
b) List all valid (X, Y) pairs.

(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

1. Consider a backtracking algorithm attempting to solve a CSP.  
   a) What happens if the algorithm chooses a value that leads to an inconsistency?

If the algorithm chooses a value that leads to an **inconsistency**, it must **backtrack** to the previous variable and try a different value.

b) Why is backtracking inefficient in some cases?

Backtracking is inefficient in some cases because it may explore many invalid paths **before finding a solution**, especially if **no inference** (like forward checking) is used to detect conflicts early.

**📌 Section C: Analytical Task – Backtracking Walkthrough**

1. Consider the following simple map coloring problem:
   * Variables: A, B, C
   * Domains: {Red, Green}
   * Constraints: A ≠ B, B ≠ C
2. Show step-by-step how backtracking search would work on this problem.

**Step 1: Assign A = Red**

**Step 2: Try B = Red**

* Constraint A ≠ B → (Red = Red) → **Backtrack**

**Step 3: Try B = Green**

* Constraint A ≠ B

**Step 4: Try C = Red**

* Constraint B ≠ C

**Final Assignment:**  
A = Red, B = Green, C = Red

b) How many assignments are tried before finding a solution?

A = Red → 1

B = Red → invalid → backtrack

B = Green → 1

C = Red → 1

**Total assignments tried: 4**

c) Write down a valid solution.

**A = Red**

**B = Green**

**C = Red**

**📌 Section D: Forward Checking (Inference)n**

1. **Explain in your own words** what forward checking is and how it helps during the CSP solving process.

**Forward checking** is a technique used during CSP solving where, **after assigning a value to a variable**, the algorithm looks ahead and **removes inconsistent values** from the domains of connected (unassigned) variables.

1. Using the same map coloring problem as in Q7:
   * Variables: A, B, C
   * Domains: {Red, Green}
   * Constraints: A ≠ B, B ≠ C
2. If A is assigned **Red**, show the remaining domains of B and C after **forward checking**.

* Constraint: **A ≠ B**  
  → B **cannot** be Red

**Updated domains:**

* B: {Green}
* C: {Red, Green} (no constraint directly between A and C)

b) Now if B is assigned **Green**, what is the domain of C after forward checking?

* Constraint: **B ≠ C**  
  → C **cannot** be Green

**Updated domain:**

* C: {Red}

1. Consider:

* Variables: X, Y, Z
* Domains: {1, 2, 3} for each
* Constraints: X ≠ Y, Y ≠ Z

If X is assigned 2:  
a) Apply forward checking to update the domains of Y and Z.

**From constraint X≠YX \neq YX=Y:**

* X = 2 → Remove 2 from Y’s domain

**Updated domains:**

* Y: {1, 3}
* Z: {1, 2, 3} (unchanged, since no direct constraint with X)

b) Which value assignment to Y would force backtracking in the next step?

Let’s test Y = 1 and Y = 3:

* If **Y = 1** → constraint Y≠ZY \neq ZY=Z → Z ≠ 1 → Z = {2, 3} ✅
* If **Y = 3** → Z ≠ 3 → Z = {1, 2} ✅

**📌 Section E: Reflection**

1. In your own words, answer:

* What is the most important benefit of using **forward checking** in CSP solving?

Forward checking reduces the search space by eliminating invalid values from future variables' domains. This helps detect conflicts early and reduces unnecessary backtracking.

Ask ChatGPT

* When might it not be sufficient on its own?

Forward checking may not be sufficient on its own when constraints involve **multiple variables** (higher-order) or **indirect relationships** between unassigned variables. In such cases, it can't detect all future conflicts.