L5

Wednesday, August 13, 2025 4:54 PM

L05 - CSP

Constraints Satisfaction Problems

Dr. Sandareka Wickramanayake

1

Reference

- Artificial Intelligence - A Modern Approach – Chapter 6 – Sections 1 and 4.

2

Constraint Satisfaction Problems (CSPs)

- In a standard search problem:
  - State is a "black box" to the search algorithm – it is not aware of the internal structure of the states.
  - Internal data structure of states can only be accessed by problem-specific functions.
    - Successor function, heuristic function, and goal test
- CSP:
  - States and goal test of a CSP conforms to a standard structure and a simple representation
    - This allows search algorithms to take advantage of the structure of states and use general-purpose heuristics instead of problem-specific ones.

3

A CSP is just a fancy name for this kind of puzzle. It has three simple parts:

1. The Things You Need to Fill In (Variables)
These are the empty squares in your Sudoku grid or the blank spaces in your crossword. In a CSP, these are called **Variables**.

2. The Choices You Can Make (Domain)
For each empty square, what can you write? In Sudoku, you can only use the numbers 1 through 9. This list of possible choices for each variable is called its **Domain**.

3. The Rules of the Puzzle (Constraints)
These are the rules you must follow.

- In Sudoku: "No two numbers in the same row can be the same."
- In a crossword: "This 5-letter word must be a type of fruit and its third letter must be 'A'."

These rules are the **Constraints**. They tell you which choices are allowed and which are not.

How is a CSP different from a "standard" search problem?

Let's compare it to a problem like finding a path on a map:

- **Standard Search (e.g., GPS navigation):**
  - The algorithm just sees states as a "black box." It knows point A and point B are connected, but it doesn't necessarily know why. It just blindly follows the connections.
  - It needs a custom-made heuristic (like "straight-line distance") designed specifically for maps.
- **Constraint Satisfaction (e.g., Sudoku):**
  - The algorithm knows the *internal* structure of the problem. It knows exactly what the variables, domains, and constraints are.
  - Because it knows the **rules**, it can use general, smart tricks to solve it faster. For example, if a square can *only* be a '3', it can put a '3' there immediately and then remove '3' as an option from all the squares in the same row and column.

Constraint Satisfaction Problems(CSPs)

- CSP is defined by
  - A set of variables $X = \{X_1, X_2, \dots, X_n\}$, where each X_i, can take values from domain D_i
  - A set of constraints, $C = \{C_1, C_2, \dots, C_m\}$
- *A* domain, D_i consists of a set of allowable values $\{v_1, v_2, \dots, v_k\}$ for variable X_i
- E.g., if X_i is Boolean the domain is *{true, false}*
- Different variables can have different domains of different sizes.

4

Constraint Satisfaction Problems(CSPs)

- Each constraint C_j involves a subset of X and specifies legal combinations of values for that subset
- A state is defined by an assignment of values to all or some of the variables, $\{X_i = v_i, X_j = v_j, \dots\}$
  - E.g., If X_1 and X_2 both have the domain {1,2,3}, then the constraint saying that X_1 must be greater than X_2 can be written as $\langle (X_1, X_2), \{(3,1), (3,2), (2,1)\} \rangle$ or $\langle (X_1, X_2), X_1 > X_2 \rangle$

Constraint Satisfaction Problems(CSPs)

- An assignment that doesn't violate any constraint is called a consistent or legal assignment.
- If every variable is assigned a value, it is a complete assignment.
- A solution to a CSP is a complete and consistent assignment.
  - E.g., One that has all variables assigned with values and satisfies all the constraints

Imagine you are a party planner assigning guests to tables.

- **Variables:** The guests (Alice, Bob, Chloe, David).
- **Domain:** The available tables (Table 1, Table 2).
- **Constraints:** The rules you must follow.
  - *Constraint 1:* Alice and Bob must sit at the **same table** (they're best friends).
  - *Constraint 2:* Chloe and David must sit at **different tables** (they just broke up).

1. Consistent (Legal) Assignment

This is an assignment that **follows all the rules (constraints)**, but it doesn't have to be finished.

- **Example:** You assign only Alice and Bob so far. You put them both at Table 1.
- **Is it consistent?** ✅ Yes! You followed the rule that they must sit together. You haven't broken any rules.

2. Complete Assignment

This is an assignment where **every single guest has been assigned to a table**. It might follow the rules, or it might not!

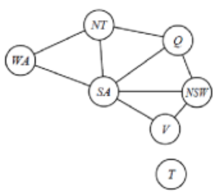
- **Example:** You assign everyone:
  - Alice -> Table 1
  - Bob -> Table 1
  - Chloe -> Table 1
  - David -> Table 1
- **Is it complete?** ✅ Yes! Everyone has a table.
- **Is it consistent?** ❌ No! You broke the rule that Chloe and David must sit at different tables.

3. Solution (Complete + Consistent Assignment)

This is the perfect outcome. Everyone is assigned to a table, and all the rules are followed.

- **Example:** You assign everyone like this:
  - Alice -> Table 1
  - Bob -> Table 1 (follows Rule 1)
  - Chloe -> Table 1
  - David -> Table 2 (follows Rule 2)
- **Is it complete?** ✅ Yes! Everyone has a table.
- **Is it consistent?** ✅ Yes! All rules are satisfied.

Example: Map-Coloring



Constraint graph

Nodes – Variables
Edges – Connect any two variables that participate in a constraint.

- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red, green, blue}
- Constraints: adjacent regions must have different colors
  - e.g., WA ≠ NT, Q ≠ NW, … etc.
  - Legal values under the constraint WA ≠ NT are;
    (WT,NT) ϵ {(red,green), (red,blue), (green,red), (green,blue), (blue,red), (blue,green)}

Example: Map-Coloring

- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green.

Why Formulate a Problem as a CSP?

- Provide a natural representation for a wide variety of problems.
- CPS solvers are fast and efficient.
- Can quickly eliminate a large portion of the search space that violates the constraints which an atomic state-space searcher cannot.
  - E.g., Once we have chosen SA = blue in the Australia problem, we can conclude that none of the five neighboring variables can take on the value.

9

- **Standard Search (Atomic Search):** This method would guess every single combination one by one: 0001, 0002, 0003... all the way up to 9999. It's slow and dumb because it wastes time on codes that obviously break the "no zero" rule, like 0001.
- **CSP Search:** This method is smart. It would immediately say, "The digit '0' is not allowed *anywhere*." It then only tries combinations from 1111 to 9999. It **eliminated a huge chunk of bad possibilities** right from the start.

Real-world CSPs

- Class Assignment problems
  - E.g., who teaches what class
- Timetabling problems
  - E.g., which class is offered when and where?
- Transportation Scheduling
- Factory Scheduling

Notice that many real-world problems involve real-valued variables

10

Variations on the CSP Formalism

- Type of variables
  - Discrete variables
    - Finite domains:
      - n variables, each having a domain of size d, leads to $O(d^n)$ possible complete assignments
      - E.g., Map coloring problems and 8-queens
    - Infinite domains:
      - Integers, strings, etc.
      - E.g., job scheduling, where variables are start/end days for each job
  - Continuous variables
    - E.g., start/end times for Hubble Space Telescope observations
    - Liner programming.

11

1. Discrete Variables (You pick from a list)

The answer is a distinct, separate item.

A) Finite Domains (The list is short)

- **What it means:** The number of choices for each variable is small and known.
- **Example 1: Map Coloring**
  - Your list (domain) is: {Red, Green, Blue}. That's it. Just 3 choices.
- **Example 2: The 8-Queens Puzzle** (placing 8 queens on a chessboard so none can attack each other)
  - For each queen, the domain is the 64 squares on the board. This is a large but *finite* number of choices.
- **The Challenge:** Even with a small list of choices per variable, the total number of possible combinations becomes **astronomically huge** very fast (`d` choices for `n` variables = `dⁿ` possible assignments). CSP techniques are brilliant at managing this explosion.

B) Infinite Domains (The list is endless)

- **What it means:** The choices are from a set that has no end, like integers or words.
- **Example: Job Scheduling**
  - **Variable:** "Start day for Job X"
  - **Domain:** Any integer from 1 to 365 (day of the year). This is a finite domain.
  - But if the job could be scheduled on any day, ever, the domain would be *all integers*, which is infinite. In practice, we often restrict it to a finite range, but the potential is infinite.

2. Continuous Variables (You pick from a range)

The answer is any number within an interval.

- **What it means:** Instead of picking from a list of separate items, you can pick any value in a smooth, unbroken range.
- **Example: Hubble Telescope Scheduling**
  - **Variable:** "Start time for observing a specific star."
  - **Domain:** Any real number (e.g., 12:47:32.145 PM) between two dates. The number of possible start times is literally infinite and continuous.
- **How to solve it?** This often requires very advanced mathematical techniques like **Linear Programming**, which is a powerful method for optimizing problems (e.g., "get the maximum observation time") with continuous variables and linear constraints.

Variations on the CSP Formalism

- Type of constraints
  - Unary constraints involving a single variable.
    - E.g., SA ≠ green
  - Binary constraints involving pairs of variables.
    - E.g., SA ≠ WA
  - Global constraints involving an arbitrary number of variables.

12

1. Unary Constraint (A Rule About ONE Thing)

- **What it is:** A rule that involves only **a single variable**. It restricts the choices for that one thing all by itself.
- **Simple Example:** Imagine a special rule for South Australia (SA): "South Australia can never be colored green."
- **What it does:** It directly removes "green" from SA's list of possible colors. Its domain becomes just {red, blue}.
- **Analogy:** A note on your shopping list that says "No bananas." It only affects the "banana" item.

2. Binary Constraint (A Rule Between TWO Things)

- **What it is:** A rule that involves **a pair of variables**. It defines a relationship between them.
- **Simple Example:** This is the most common type in the map problem. "Western Australia (WA) and Northern Territory (NT) must have different colors." (WA ≠ NT)
- **What it does:** It doesn't say what color WA or NT must be individually. It only says that whatever they are, they can't be the same.
- **Analogy:** A rule on your shopping list: "If I buy ice cream, I must also buy chocolate syrup." The rule connects two items.

3. Global Constraint (A Big Rule About MANY Things)

- **What it is:** A rule that involves **an arbitrary number of variables** (often more than two). It's a complex rule that applies to a whole group.
- **Simple Example:** Imagine a new rule: "**At least three** of the seven states must be colored red." This rule involves all variables at once.
- **Another Classic Example (AllDifferent):** "All five of these states must all have different colors." This is a very common global constraint.
- **What it does:** It's a more complex condition that can't be broken down into just simple pairs. It looks at the entire assignment.
- **Analogy:** A rule on your shopping list: "The total cost of all items must be less than $50." This rule involves every single item you choose.

Local Search for CSPs

- Hill-climbing, simulated annealing, and others can be used for CSPs
  - Typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - Allow states with unsatisfied constraints
  - Operators reassign variable values

- Initial state: Some assignment to all variables. E.g., random
- Successor function: Usually changes the value of a single variable
- Variable selection: Randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - Choose a value that violates the fewest constraints
    - E.g., hill-climb with $h(n)$ = total number of violated constraints

13

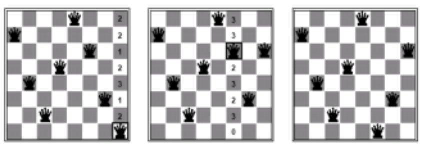
Imagine you're trying to solve a big, messy puzzle where many pieces are in the wrong place. Instead of starting from scratch, you take the puzzle as it is and try to fix it one piece at a time. This is what **local search** does for Constraint Satisfaction Problems (like scheduling, puzzles, etc.).

Step-by-Step Simple Explanation

1. **Start with a Complete (but Messy) Guess**
   - Don't leave any variable blank. Just assign every variable a random value to create a full, but probably incorrect, solution.
   - *Example: For a class schedule, randomly assign teachers, classes, and rooms to all time slots. It will be full of conflicts (e.g., two classes in the same room), but it's a complete starting point.*
2. **Find a Problem**
   - Look at your messy solution and find a variable that is causing trouble (a "conflicted variable").
   - *Example: Find a teacher who is scheduled to teach two classes at the same time.*
3. **Fix That One Problem the Best You Can**
   - This is the **min-conflicts heuristic**. For the problematic variable you found, try changing its value. Look at all the possible new values you could assign it and choose the one that causes the **fewest number of new problems.**
   - *Example: For the double-booked teacher, look at all the other available time slots. Move the teacher to the slot that causes the fewest other scheduling conflicts.*
4. **Repeat**
   - Keep doing steps 2 and 3: find a conflicted variable and fix it by choosing the value that causes the fewest conflicts.
   - You are essentially "hill-climbing" by always trying to reduce the total number of errors in your solution.

Min-conflicts Example

- A two-step solution for an 8-queens problem using min-conflicts heuristic
- At each stage a queen is chosen for reassignment in its column
- The algorithm moves the queen to the min-conflict square, breaking ties randomly.



14

Local Search for CSPs

```
function MIN-CONFLICTS(csp, max_steps) return solution
  or failure
 inputs: csp (a constraint satisfaction problem),
    max_steps (the number of steps allowed before
 giving up)
 current ←  an initial complete assignment for csp
 for i = 1 to max_steps do
    if current is a solution for csp
    then return current
    var ←  a randomly chosen, conflicted variable
 from VARIABLES[csp]
    value  ←  the value v for var that minimize
 CONFLICTS(var,v,current,csp)
    set var = value in current
 return failure
```

15

1. **Start** with a complete but broken solution.
2. **Find** a broken part.
3. **Fix** that part in the way that causes the least amount of *new* breakage.
4. **Repeat** until everything is fixed or you run out of time.