Unit 3:Adversarial Search and Games

Types of Constraints

Constraint Satisfaction Problem

- The constraint satisfaction problem consists of three components:

i) X- set of Variables

ii) D- Set of Domains (one for each variable)

iii) C- set of constraints

- A constraint satisfaction problem (CSP) is a problem where variables must be assigned values that satisfy certain constraints.

Examples of Constraint satisfaction Problems

- **n-queen problem:** In n-queen problem, the constraint is that no queen should be placed either diagonally, in the same row or column.
- **Cryptarithmetic Problem:** This problem has one most important constraint that is, we cannot assign a different digit to the same character. All digits should contain a unique alphabet.
- **Sudoku:** every row, column and 3* 3 board should have unique digit.
- **Graph/map coloring problem:** no two adjacent region have same colour.

Constraint satisfaction Problems

An assignment of values to a variable can be done in three ways:

- **Consistent or Legal Assignment:** An assignment which does not violate any constraint or rule is called Consistent or legal assignment.
- **Complete Assignment:** An assignment where every variable is assigned with a value, and the solution to the CSP remains consistent. Such assignment is known as Complete assignment.
- **Partial Assignment:** An assignment which assigns values to some of the variables only. Such type of assignments are called Partial assignments.

Types of Constraints in CSP

Constraint Types in CSP

With respect to the variables, basically there are following types of constraints:

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.
- **Binary Constraints:** It is the constraint type which relates two variables. A value **x2** will contain a value which lies between **x1** and **x3**.
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

Constraint Propagation in CSP

Constraint Propagation

- Constraint Propagation is a technique used in **Constraint Satisfaction Problems (CSPs)** to reduce the search space by enforcing constraints before or during the search. It systematically eliminates inconsistent values from variable domains by applying constraints iteratively.

- **How Constraint Propagation Works**
- Each variable in a CSP has a domain of possible values.
- Constraints restrict which values can be assigned to variables.
- Constraint propagation reduces domains by eliminating values that violate constraints, making the search process more efficient.

Benefits of Constraint Propagation

- Reduces the search space by eliminating inconsistent values early.
- Helps avoid unnecessary backtracking in search algorithms.
- Improves efficiency, especially in large CSPs.

Applications of Constraint Propagation

- **Sudoku Solving** (removing invalid numbers from each row, column, and block).
- **Scheduling Problems** (ensuring time slots do not overlap).
- **Graph Coloring** (ensuring no two adjacent nodes have the same color).
- **AI Planning** (optimizing task assignments and dependencies).

Types of consistencies

There are following local consistencies which are discussed below:

- **Node Consistency:** A single variable is said to be **node consistent i**f all the values in the variable's domain satisfy the **unary constraints** on the variables.
- **Arc Consistency:** A variable is arc consistent if every value in its domain satisfies the **binary constraints** of the variables.
- **Path Consistency:** When the evaluation of a s**et of two variable with respect to a third variable** can be extended over another variable, satisfying all the binary constraints. It is similar to arc consistency.

1. Node Consistency (1-Consistency)

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

Definition:

A variable X is **node-consistent** if for every value $v \in D(X)$, v satisfies all **unary constraints** on X.

Example:

Consider a variable X with domain $D(X) = \{1, 2, 3, 4, 5\}$ and a unary constraint $X > 2$.

- **Node Consistency Check:** Remove values **1** and **2** since they do not satisfy $X > 2$.
- **Resulting Domain:** $D(X) = \{3, 4, 5\}$

2. Arc Consistency (2-Consistency)

An **arc** (directed edge) between two variables is **arc-consistent** if **for every value of one variable, there exists a valid value in the other variable's domain that satisfies the constraint.**

Definition:

A constraint between two variables X and Y is **arc-consistent** if:

$$\forall x \in D(X), \exists y \in D(Y) \text{ such that } (x, y) \text{ satisfies the constraint on } (X, Y).$$

Example:

Suppose we have variables:

- X with domain $\{1, 2, 3\}$
- Y with domain $\{1, 2, 3\}$
- Constraint: $X < Y$

Checking Arc Consistency:

1. For $X = 1$ → valid values in Y are $\{2, 3\}$
2. For $X = 2$ → valid value in Y is $\{3\}$
3. For $X = 3$ → no valid value in Y Remove 3 from $D(X)$)

Updated Domains:

- $D(X) = \{1, 2\}$
- $D(Y) = \{1, 2, 3\}$

Arc consistency can be enforced using the AC-3 algorithm.

3. Path Consistency (3-Consistency)

A **path** involving three variables is **path-consistent** if for every valid assignment to two variables, there exists a valid assignment for the third variable.

Definition:

A CSP is **path-consistent** if for every pair of variables X and Y, and for every value (x, y) satisfying the binary constraint between them, there exists a value z in $D(Z)$ such that (x, z) and (y, z) satisfy the constraints.

Example:

Consider variables X, Y, Z with domains:

- $D(X) = \{1, 2\}$
- $D(Y) = \{2, 3\}$
- $D(Z) = \{3, 4\}$

Constraints:

- $X < Y$
- $Y < Z$

Checking Path Consistency:

1. $(X = 1, Y = 2)$ must have $Z > 2$ (valid values in Z are $\{3, 4\}$)
2. $(X = 2, Y = 3)$ must have $Z > 3$ (valid value in Z is $\{4\}$)

Since for every pair of values, there exists a value in the third variable satisfying the constraints, the problem is **path-consistent**.

Local Consistency Levels

Nogood: forbidden tuple of values

- In the initial constraints
- Discovered during search / local consistency process

Local consistency levels:		nogood size	
• Node consistency:	1-consistency	0	
• Arc consistency:	2-consistency	1	*removes values from domains*
• Path consistency:	3-consistency	2	*discovers forbidden value pairs*
• …..			
• K-consistency:		K-1	*discovers forbidden value combinations*

Constraint Satisfaction: Propagation

Node consistency

Node consistency requires that every **unary constraint** on a variable is **satisfied by all values in the domain of the variable**, and vice versa. This condition can be trivially enforced by reducing the domain of each variable to the values that satisfy all unary constraints on that variable.

For example, given a variable $\{V\}$
with a domain of $\{1,2,3,4\}$
and a constraint $\{V<3\}$
Node consistency would restrict the domain to $\{1,2\}$ and the constraint could then be discarded.

Node Consistency (NC)

NC:

- Variable x_i is *node consistent* iff every value of D_i is allowed by R_i
- *P* is NC iff every variable is NC

Unitary constraint on x_i

NC Algorithm:

```
procedure NC-1 (X,D,C)
  for all xi∈X do
    for all a∈ Di do
      if a∉ Ri then Di:= Di-{a};
```

$D_i := D_i \wedge R_i$

i: 1, …, n

Constraint Propagation:Inference in CSPs

Arc Consistency

- The pair (X, Y) of constraint variables is arc consistent if for each value
  $x \in DX$ there exists a value $y \in DY$ such that the assignments X = x and Y = y satisfy all binary constraints between X and Y. A CSP is a**rc consistent if all variable pairs are arc consistent.**
- Consider a simple CSP with the variables A and B subject to their respective domains DA={1,2} and DB={1,2,3} , as well as the binary constraint $A < B$. We see that value 1 can be safely removed

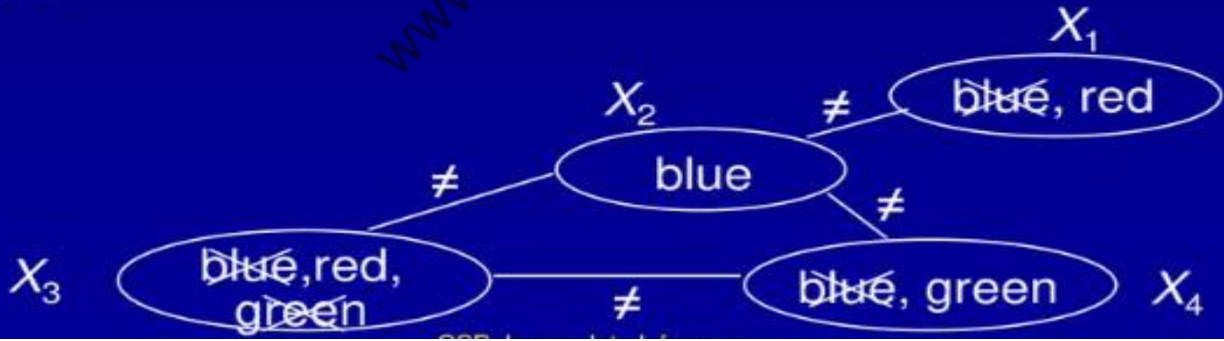
Filtering by Arc Consistency

If for $a \in D_i$ there not exists $b \in D_j$ such that $(a, b) \in R_{ij}$, a can be removed from D_i (a will not be in any sol)

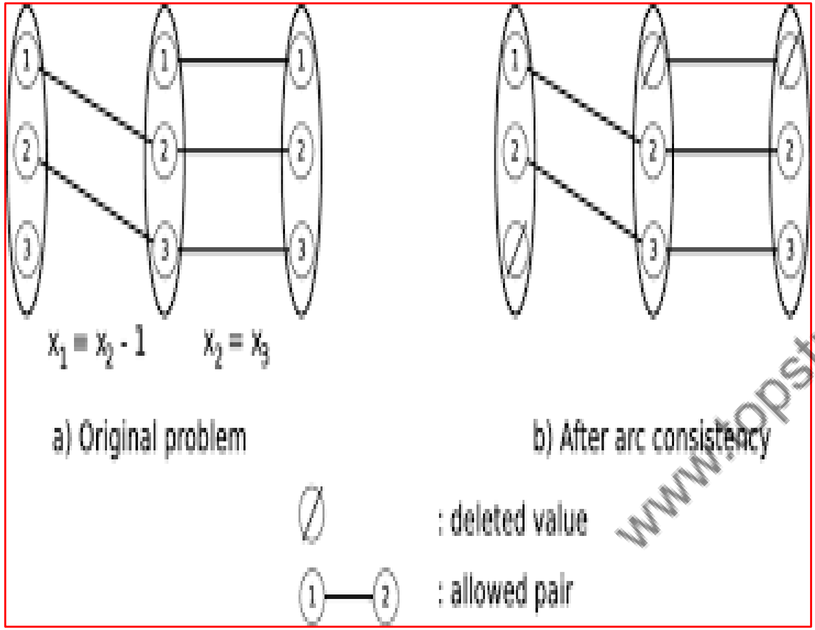
Domain filtering:

- Remove arc-inconsistent values
- Until no changes

Example:



Arc Consistency



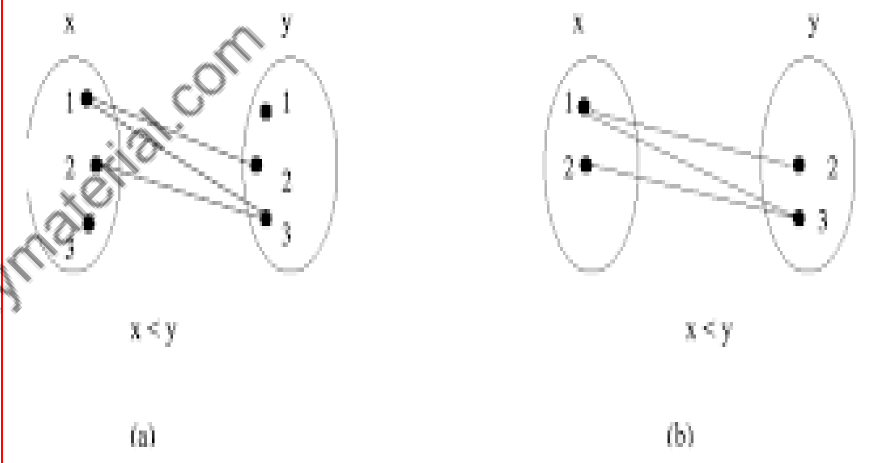


Figure 3.1: A matching diagram describing the arc-consistency of two variables x and y. In (a) the variables are not arc-consistent. In (b) the domains have been reduced, and the variables are now arc-consistent.

Revise for arc-consistency

REVISE$((x_i), x_j)$

input: a subnetwork defined by two variables $X = \{x_i, x_j\}$, a distinguished variable x_i, domains: D_i and D_j, and constraint R_{ij}

output: D_i, such that, x_i arc-consistent relative to x_j

```
for each a_i ∈ D_i
    if there is no a_j ∈ D_j such that (a_i, a_j) ∈ R_ij
        then delete a_i from D_i
    endif
endfor
```

Figure 3.2: The Revise procedure

Path-consistency

Definition 3.3.2 (Path-consistency) *Given a constraint network* $\mathcal{R} = (X, D, C)$, *a two variable set* $\{x_i, x_j\}$ *is path-consistent relative to variable* x_k *if and only if for every consistent assignment* $(< x_i, a_i >, < x_j, a_j >)$ *there is a value* $a_k \in D_k$ *s.t. the assignment* $(< x_i, a_i >, < x_k, a_k >)$ *is consistent and* $(< x_k, a_k >, < x_j, a_j >)$ *is consistent. Alternatively, a binary constraint* R_{ij} *is path-consistent relative to* x_k *iff for every pair* $(a_i, a_j), \in R_{ij}$, *where* a_i *and* a_j *are from their respective domains, there is a value* $a_k \in D_k$ *s.t.* $(a_i, a_k) \in R_{ik}$ *and* $(a_k, a_j) \in R_{kj}$. *A subnetwork over three variables* $\{x_i, x_j, x_k\}$ *is path-consistent iff for any permutation of* (i, j, k), R_{ij} *is path consistent relative to* x_k. *A network is path-consistent iff for every* R_{ij} *(including universal binary relations) and for every* $k \neq i, j$ R_{ij} *is path-consistent relative to* x_k.

Path-consistency

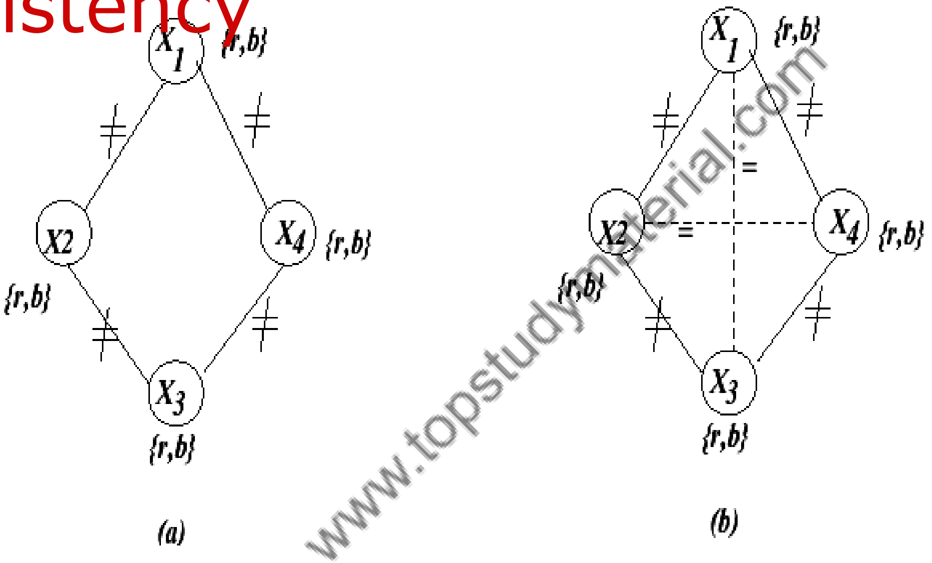
REVISE-3($(x, y), z$)

input: a three-variable subnetwork over (x, y, z), R_{xy}, R_{yz}, R_{xz}.
output: revised R_{xy} path-consistent with z.

```
1.  for each pair (a, b) ∈ R_xy
2.        if no value c ∈ D_z exists such that (a, c) ∈ R_xz and (b, c) ∈ R_yz
3.              then delete (a, b) from R_xy.
4.        endif
5.  endfor
```

Figure 3.9: Revise-3

Example: before and after path-consistency



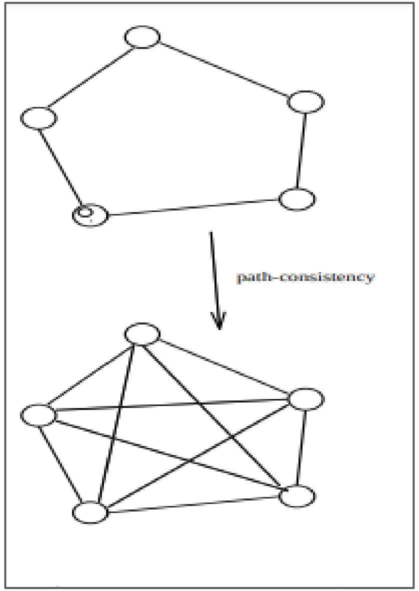


Figure 3.12: A graph-coloring graph (a) before path-consistency (b) after path-consistency

Backtracking Search

N Queen

Gap coloring