Constraint Satisfaction Problem

Waliul Islam Sumon

CSE 4109: Artificial Intelligence

Courtesy of: Md Mehrab Hossain Opi

Introduction

- In traditional search we treated each state as black box with no internal structure.
  - Each state was atomic or indivisible.
- Today we will use a factored representation for each state.
  - A set of variables, each of which has a value.
- A problem is solved when each variable has a value that satisfies all the constraints on the variable.
- A problem described this way is called a constraint satisfaction problem or CSP.

Definition

- A constraint satisfaction problem consists of three components.

X is a set of variables, $\{X_1, X_2, \ldots, X_n\}$.

D is a set of domains, $\{D_1, D_2, \ldots, D_n\}$.

C is a set of constraints that specify allowable combinations of values.

Definition

- Each domain D_i consists of a set of allowable values, $\{v_1, v_2, \ldots, v_k\}$ for variable X_i.

- Each constraint C_i consists of a pair $(scope, rel)$, where $scope$ is a tuple of variables that participate in the constraint and rel is a relation that defines the values that those variables can take on.

- A relation can be represented as an explicit list of all tuples of values that satisfy the constraint.

Definition

- To solve a CSP, we need to define a state space and the notion of a solution.
- Each state in a CSP is defined by an **assignment** of values to some or all of the variables, $\{X_i = v_i,\ X_j = v_j, \ldots\}$.
- An assignment that does not violate any constraints is called a ***consistent or legal assignment*.**
- A ***complete assignment*** is one in which every variable is assigned.
- A solution to a CSP is a ***consistent, complete assignment**.*
- A partial assignment is one that assigns values to only some of the variables.

Map Coloring Problem

- Consider the following map
- We need to color each region either *red, green, or blue* in such a way that no neighboring regions have the same color.

Map Coloring Problem

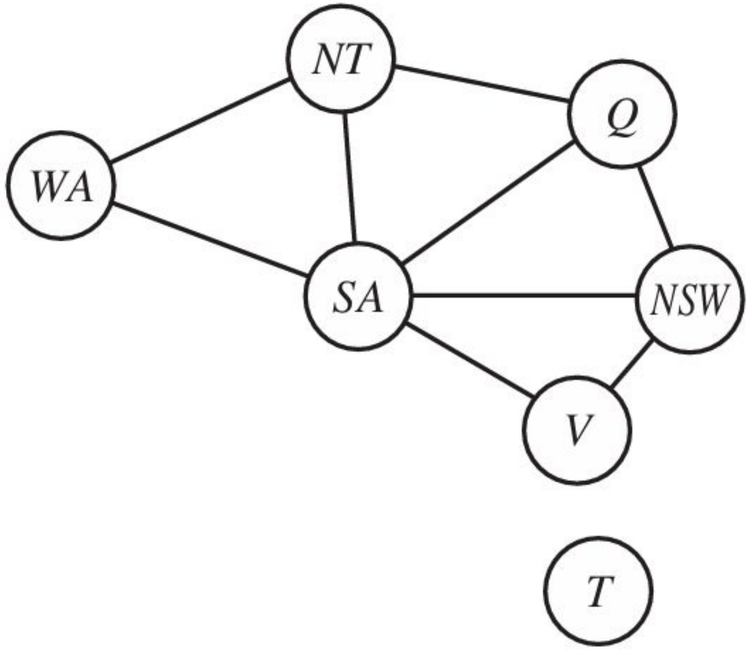
- To formulate this as a CSP, we define the variables to be the region.
  $X = \left\{ \begin{matrix} WA, NT, Q, \\ NSW, V, SA, T \end{matrix} \right\}.$
- The domain of each variable is the set $D_i = \{red, green, blue\}$.

Map Coloring Problem

- The constraints require neighboring regions to have distinct colors.
- There are nine places where regions have border.
- We get nine constraints.

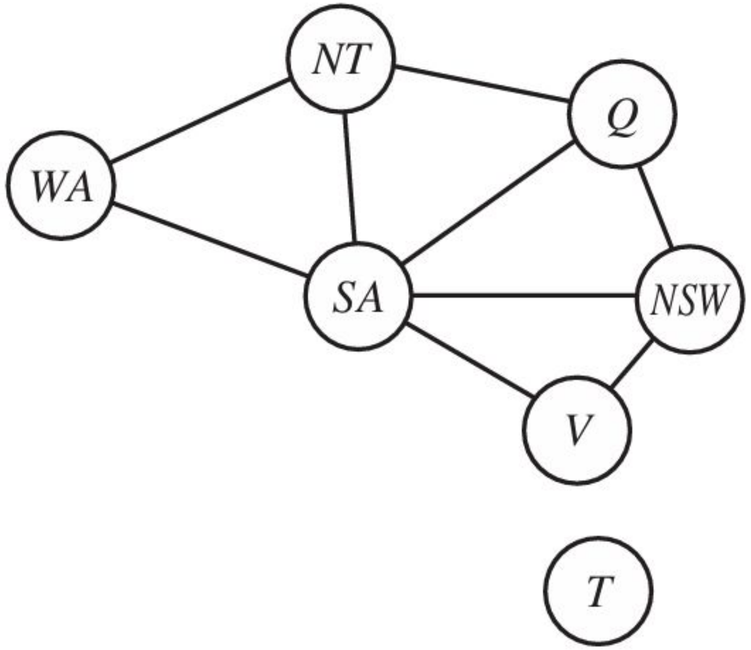
Map Coloring Problem

- $$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$



Map Coloring Problem

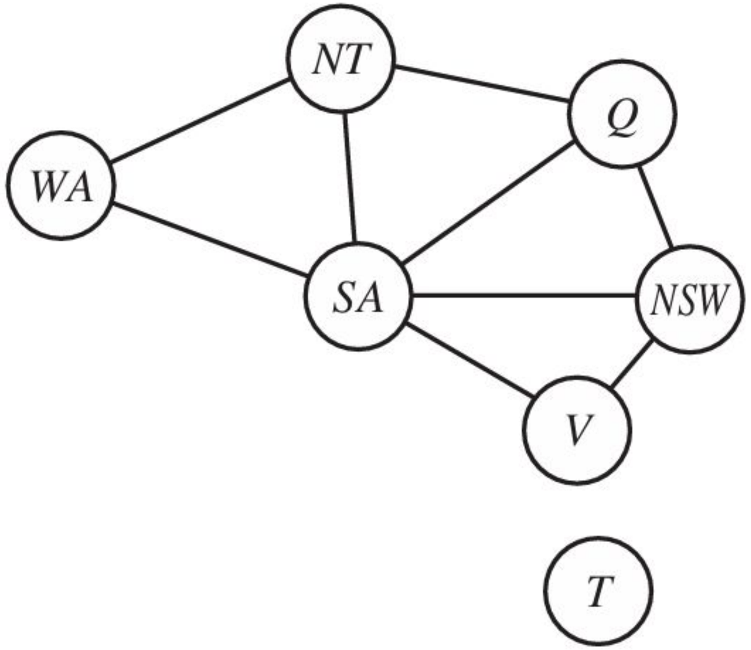
- $SA \neq WA$ is the shortcut for $\langle (SA, WA), SA \neq WA \rangle$.



Map Coloring Problem

- One possible solution is

$$\{WA = red, NT = green\ , Q = red, NSW = green, V = red, SA = blue, T = red\}$$

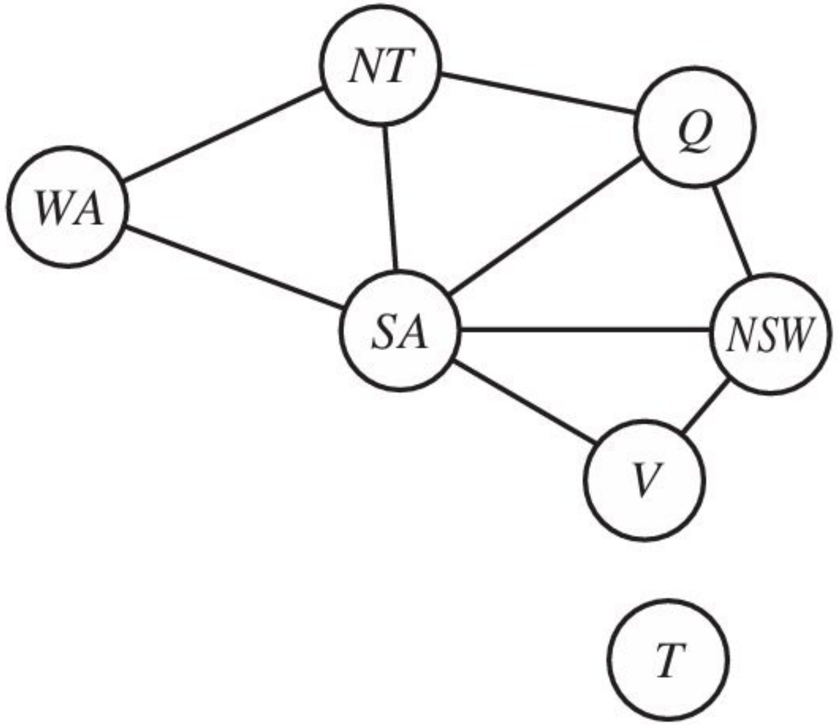


Map Coloring Problem

We can visualize a CSP as a constraint graph.

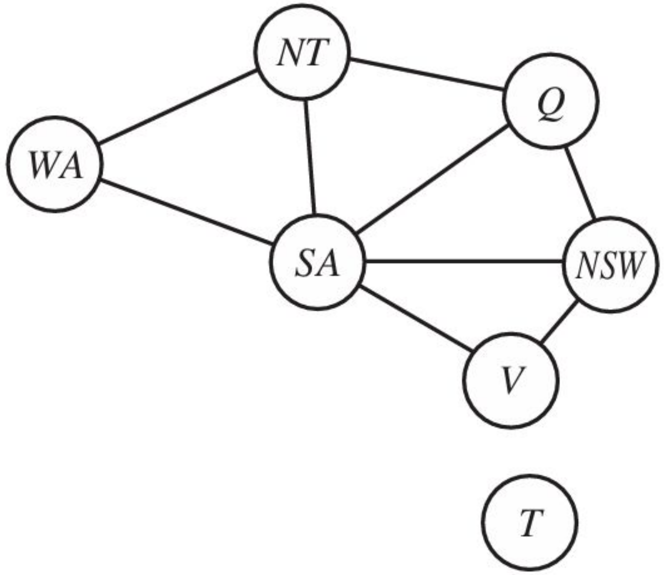
The nodes correspond to variables.

The edges connects any two variable that participate in a constraint.



Why CSP?

- Faster than state-space searchers.
- Quickly eliminates a large portion of search space.
- For example, If $SA = blue$, then all 5 $neighbors \neq blue$.
- Without constraint propagation: $3^5 = 243$ possibilities.
- With propagation (no blue allowed): $2^5 = 32$.
- 87% fewer assignments.



Job-shop Scheduling

- Consider the problem of scheduling the assembly of a car.
- The whole job is composed of tasks.
- We can model each task as a variable.
- Value of each variable is the time when the task starts.
  - Expressed as an integer number of minutes.

Job-shop Scheduling

- Constraints can assert that one task must occur before another.
- Only so many tasks can go on at once.
- Also specify that a task takes a certain amount of time to complete.

Job-shop Scheduling

- Consider a small part of the car assembly.
- Consists of 15 task.
  - Install axles (front and back)
  - Affix all four wheels (right and left, front and back)
  - Tighten nuts for each wheel
  - Affix hubcaps
  - Inspect the final assembly.
- Using variables:

$$X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\}$$

Job-shop Scheduling

- Value of each variable is the time that the task starts.
- Now we can add precedence constraints.
- Whenever a task T_1 must occur before task T_2, and task T_1 takes duration d_1 to complete, we add an arithmetic constraint

$$T_1 + d \leq T_2$$

Job-shop Scheduling

- Suppose we have four workers to install wheels, but they have to share one tool that helps put the axle in place.

- We need a disjunctive constraints to say that $Axle_F$ and $Axle_B$ must not overlap in time; either one comes first or the other does:

$$(Axle_F + 10 \leq Axle_B) \; or \; (Axle_B + 10 \leq Axle_F)$$

\- Combines logic and arithmetic.

Job-Shop Scheduling

- We also say that inspection comes last and takes 3 minutes.
- For every variable except $Inspect$ we add a constraint of the form $X + d_x \leq Inspect$.
- Finally, suppose there is a requirement to get the whole assembly done in 30 minutes.
- We can limit the domain of all variables to:
$$D_i = \{1, 2, 3, \dots, 27\}$$
- Our target is to assign a value to each variable to satisfy all these conditions.

Inference in CSP

- In regular state-space search, an algorithm can do only one thing. – **search**.
- In CSP there is a choice.
  - Search.
    - Choose a new variable assignment from several possibilities.
  - Do a specific type of inference called **constraint propagation**.
- Constraint propagation can run before or during search.
- Sometimes it even solves the problem entirely.

Local Consistency

- Local consistency prunes inconsistent values from the graph.
- Variables are represented as nodes, and binary constraints as arcs.
- Enforcing consistency checks each part of the graph to eliminate invalid values.
- There are different types of local consistency, like node, arc, and path consistency.
- This process reduces the search space, sometimes dramatically.

Node Consistency

- A variable is node-consistent if all its values satisfy unary constraints.
- Example: SA = {red, green, blue},
- But SA dislikes green → reduced to {red, blue}.
- A network is node-consistent if every variable is node-consistent.
- Running node consistency eliminates all unary constraints.

Arc Consistency

- X_i is **arc-consistent** with X_j if every value in X_i's domain has a matching value in X_j's domain satisfying the binary constraint.
- A network is arc-consistent if all variables are arc-consistent with their neighbors.
- Example: Constraint $Y = X^2$, domains $\{0,1,2,3,4,9\}$
- After arc consistency: $X \rightarrow \{0,1,2,3\}, Y \rightarrow \{0,1,4,9\}$
- Some problems (like map-coloring) may see **no domain reduction**.

Arc Consistency – AC-3 Algorithm

function AC-3(csp) **returns** false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components $(X,\ D,\ C)$
  local variables: $queue$, a queue of arcs, initially all the arcs in csp

  while $queue$ is not empty **do**
    $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST($queue$)
    if REVISE(csp, $X_i,\ X_j$) **then**
      if size of $D_i\ =\ 0$ **then return** $false$
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to $queue$
  return $true$

---

function REVISE(csp, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  $revised \leftarrow false$
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      $revised \leftarrow true$
  return $revised$

Arc Consistency – AC-3 Algorithm

- Assume a CSP with n variables, each with domain size at most d, and with c binary constraints (arcs).
- Each arc can be inserted in the queue only d times.
  - X_i has at most d values to delete.
- Checking consistency can be done in $O(d^2)$ time.
- Hence, we get $O(cd^3)$ total worst-case time.

Path Consistency

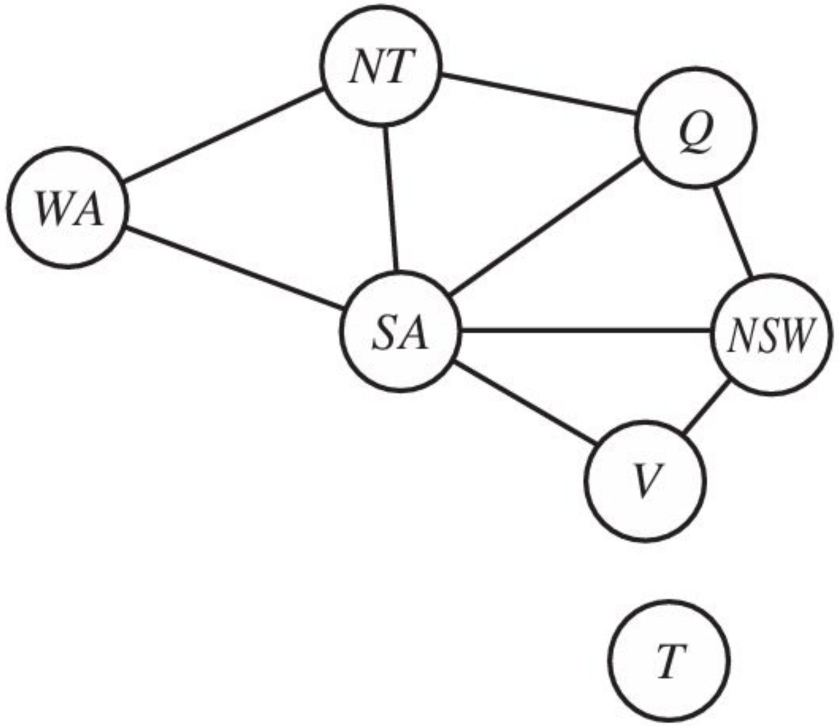
- Arc consistency reduces domains and can sometimes:
  - Solve the CSP (all domains size 1)
  - Detect failure (some domain size 0)
- Some networks remain unresolved:
  - Australia map-coloring with 2 colors
  - Every variable is already arc-consistent
  - WA, NT, SA all touch → need ≥3 colors

Path Consistency

- Arc consistency uses binary constraints; path consistency uses triples of variables.
- A pair $\{X_i, X_j\}$ is path-consistent w.r.t X_m if:
  - For every consistent assignment to $\{X_i, X_j\}$, there exists a value for X_m satisfying both $\{X_i, X_m\}$ and $\{X_m, X_j\}$ constraints.
- Think of it as looking at a “path” from $X_i \rightarrow X_m \rightarrow X_j$ to tighten domains further.

Path Consistency

- We will make the set $\{WA, SA\}$ path consistent w.r.t NT.
- Consider there are only two colors: red, blue.
- So there are two valid arc: $\{WA = red, SA = blue\}$ or $\{\mathrm{WA} = \mathrm{blue}, \mathrm{SA} = \mathrm{red}\}$
- Using two colors there are no valid assignment left for NT.



K-consistency

- A CSP is k-consistent if, for any set of $k - 1$ variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
- 1-consistency – node consistency.
- 2-consistency – arc consistency.
- 3-consistency – path consistency.

K-consistency

- A CSP is strongly k-consistent if it is k-consistent and is also $(k-1)$ consistent, $(k-2)$-consistent, … and all the way down to 1-consistent.

Global Constraints

- Global constraint is one involving an arbitrary number of variables.
  - Not necessarily all variables.
- $Alldiff$ constraint says that all the variables involved must have distinct values.

Sudoku Example

- A sudoku board consists of 81 squares.
- Initially some squares are filled with digits from 1 to 9.
- You need to fill in all the remaining squares such that no digit appears twice in any row, column, or 3×3 box.
- How can you formulate this problem in CSP?

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Sudoku Example

- There are 81 variables – one for each square.
- We use the variable names $A1 \ldots I9$.
- Domain of each empty squares $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Prefilled squares have a domain consisting of a single value.

Sudoku Example

- There are 27 $AllDiff$ constraints: one for each row, column and box of 9 squares.
  $Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)$
  ...
  $Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)$
  ...
  $Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)$

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Sudoku Example

- We can convert $Alldiff$ constraints into binary constraints. $A1 \neq A2$
- Then we can apply $AC - 3$.
- Consider $E6$.
- Using box constraint remove $\{1, 2, 7, 8\}$
- Using column constraint $\{5\ , 6, 9, 3\}$
- So, we get only $\{4\}$ – the answer.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Sudoku Example

- Now consider $I6$.
- We can remove
  - $\{2, 3, 4, 5, 6, 8, 9\}$ – using column constraint.
    - We already solved $E6$.
  - $\{1\}$ – box/row constraint.
- We get $\{7\}$ for $I6$ – the answer.
- From arc consistency we get $A6 = 1$.
- Inference continues and can solve this puzzle without any search algorithm.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Sudoku Example

- Of course, not all sudoku example can be solved using arc consistency.
- Slightly harder ones can be solved using $PC - 2$.
- To solve the hardest, we need better strategy.

Backtracking Search for CSPs

- We can apply standard DFS.
- A state would be a partial assignment.
- An action would be adding $var = value$ to the assignment.
- At each step we will consider only a single variable.
  - Why?

Backtracking Search for CSPs

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
   return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
         add {var = value} to assignment
         inferences ← INFERENCE(csp, var, value)
         if inferences ≠ failure then
            add inferences to assignment
            result ← BACKTRACK(assignment, csp)
            if result ≠ failure then
               return result
      remove {var = value} and inferences from assignment
   return failure
```

Backtracking Search for CSPs

- Earlier we used heuristic functions derived from our knowledge.
- We can solve CSPs efficiently without such domain-specific knowledge.
- Instead, we can add some sophistication to the unspecified functions.

Backtracking Search for CSPs

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, value)
        if inferences ≠ failure then
           add inferences to assignment
           result ← BACKTRACK(assignment, csp)
           if result ≠ failure then
              return result
     remove {var = value} and inferences from assignment
  return failure
```

How to Avoid Repeated Failure?

Variable and Value Ordering

- $var \leftarrow SELECT - UNASSIGNED - VARIABLE(csp)$
- Simplest strategy is to select the next unassigned variable in order, $\{X_1, X_2, \ldots\}$
  - Seldomly produces efficient search.
- Another option is to choose the variable with the fewest legal values.
  - Minimum Remaining Values (MRV) heuristic.
  - Also called “most constrained variable” or “fail-first” heuristic.

Variable and Value Ordering

- $var \leftarrow SELECT - UNASSIGNED - VARIABLE(csp)$
- But how do we select the first variable?
- We can use the degree heuristic.
- Select the variable that is involved in the largest number of constraints on other unassigned variable.
- Degree heuristic is normally used as tie-breaker.

Variable and Value Ordering

- Once a variable is chosen, we need to select the order of value to assign.
- We can use the least-constraining –value heuristic.
    - Use the value that rules out the fewest choices for neighbouring variables.

- Why should variable selection be fail-first, but value selection be fail-last?

Interleaving search and inference

- One of the simplest forms of inference in called **forward checking.**
- Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it.
  - For any node Y connected with node X, delete from Y's domain any value that is inconsistent with the value chosen for X.

Interleaving search and inference

- Although forward checking detects many inconsistencies, it does not detect all of them.
  - It only makes the current variable arc consistent but not all of them.
- Maintaining Arc Consistency (MAC) detects this inconsistency.
- After a variable X_i is assigned a value, the INFERENCE procedure calls AC-3.
  - Instead of a queue of all arcs in the CSP, we start with only the arcs (X_j, X_i) for all X_j that are unassigned variables that are neighbours of X_i.

Intelligent Backtracking

- In the algorithm we have learned, whenever a branch failed we backed up to the preceding variable and tried a different value for it.
- This is called **chronological backtracking**.

Intelligent Backtracking

- A more intelligent approach is to backtrack to a variable that might fix the problem.
  - A variable that was responsible.
- We will keep track of a set of assignments.
- The backjumping method backtracks to the most recent assignment in the conflict set.
- How do we get this set?

Intelligent Backtracking

- We don’t need extra works to get the set.

Thank You.