Course 16 :198 :440 : Introduction To Artificial Intelligence
Lecture 6

Solving Problems by Searching: Constraint Satisfaction Problems

Abdeslam Boularias

Wednesday, February 19, 2025

Outline

We consider problems where a state is defined as a set of constraints. We show that treating states as more than just black boxes leads to an improved performance of search algorithms.

1. Definitions and examples
2. Inference in Constraint Satisfaction Problems (CSPs)
3. Backtracking search for CSPs.

Constraint Satisfaction Problems (CSPs)

We consider problems where a state is defined as a set of constraints. We show that treating states as more than just black boxes leads to an improved performance of search algorithms.

A constraint satisfaction problem consists of three components

- **X** : a set of variables $\{X_1, \ldots, X_n\}$.
- **D** : a set of domains $\{D_1, \ldots, D_n\}$, where D_i is the domain of variable X_i (i.e. $X_i \in D_i$).
- **C** : a set of constraints that specify which combinations of values are allowed.

Constraint Satisfaction Problems (CSPs)

To solve a CSP, we need to define the state space

- **State** : an assignment of values to some or all of the variables.
- **Consistent assignment** : an assignment that does not violate any constraints.
- **Partial assignment** : an assignment to some of the variables.
- **Complete assignment** : an assignment to all of the variables.
- **Solution** : a consistent complete assignment.

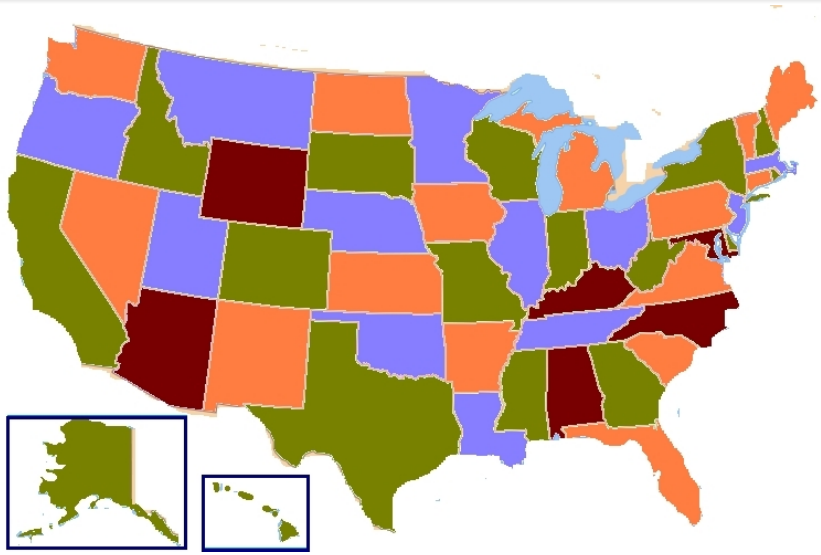
Example of CSPs : Map coloring

Four color theorem (Appel and Haken, 1976)

Given any separation of a plane into contiguous regions, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color.

Problem

Given a map and a set of colors, assign a color to each region so that no two adjacent regions have the same color.



A four-coloring of a map of the states of the United States

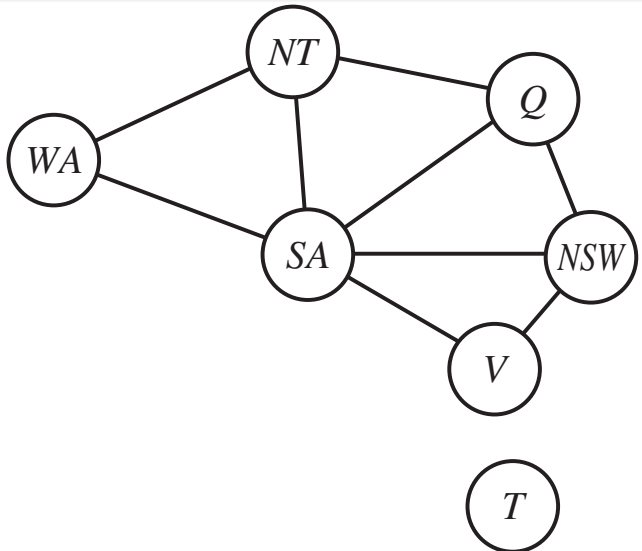
Example

Example : coloring the map of Australia

- $X = \{WA, NT, Q, NSW, V, SA, T\}$.
- $D_i = \{red, green, blue\}$.
- $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$.

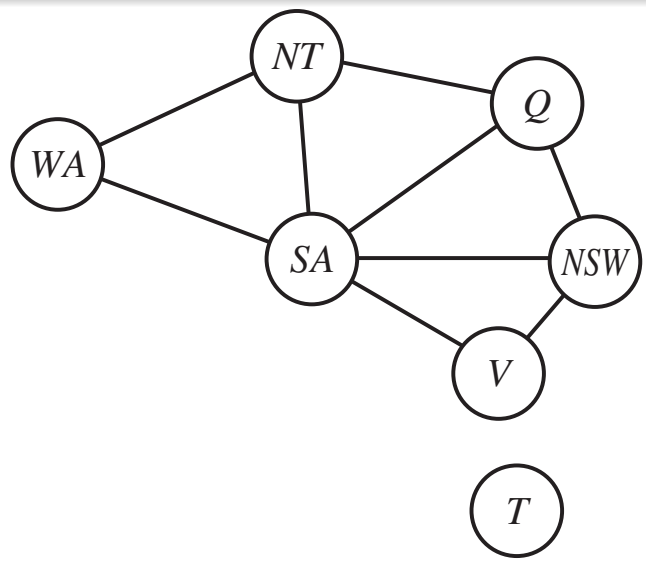
Constraint graphs are helpful for visualizing a CSP

- Nodes of the graph correspond to variables of the problem.
- A link connects any two variables that participate in a constraint.



Why formulate a problem as a CSP ?

- CSPs yield a natural representation for a wide variety of problems, and there are several CSP solvers that can be used directly.
- CSP solvers can be faster than state space searchers because they can quickly eliminate large parts of the search space that violate *some* of the constraints.
- Example : After setting $SA = \{blue\}$, we immediately eliminate all states where $blue$ is assigned to WA, NT, Q, NSW,or V. Consequently, we reduce the number of assignments from 3^5 to 2^5 (i.e. from 243 to 32)



Car assembly scheduling

Consider the problem of assembling a car, consisting of 15 tasks that must be performed in the following order :

1. install axles : front ($Axle_F$) and back ($Axle_B$), 10 minutes each,
2. affix all four wheels : right and left, front and back ($Wheel_{RF}$,$Wheel_{LF}$,$Wheel_{RB}$,$Wheel_{LB}$), 1 minute each,
3. tighten nuts for each wheel ($Nuts_{RF}$,$Nuts_{LF}$,$Nuts_{RB}$,$Nuts_{LB}$), 2 minutes each,
4. affix hubcaps (Cup_{RF},Cup_{LF},Cup_{RB},Cup_{LB}), 1 minute each,
5. and inspect the final assembly ($Inspect$), which takes 3 minutes.

Problem

Find when each task should be started in the time interval $[0, 30]$ minutes.

CSP formalization

- We represent the tasks with 15 variables :
  $X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cup_{RF}, Cup_{LF}, Cup_{RB}, Cup_{LB}, Inspect\}$.
- Each variable indicates the starting time of the corresponding task.
- The domain of each variable is $D_i = [0, 27]$.

CSP formalization

We have the following *precedence constraints*

$$
\begin{aligned}
Axle_F + 10 &\leq Wheel_{RF}; & Axle_F + 10 &\leq Wheel_{LF};\\
Axle_B + 10 &\leq Wheel_{RB}; & Axle_B + 10 &\leq Wheel_{LB};\\
Wheel_{RF} + 1 &\leq Nuts_{RF}; & Nuts_{RF} + 2 &\leq Cap_{RF};\\
Wheel_{LF} + 1 &\leq Nuts_{LF}; & Nuts_{LF} + 2 &\leq Cap_{LF};\\
Wheel_{RB} + 1 &\leq Nuts_{RB}; & Nuts_{RB} + 2 &\leq Cap_{RB};\\
Wheel_{LB} + 1 &\leq Nuts_{LB}; & Nuts_{LB} + 2 &\leq Cap_{LB}.
\end{aligned}
$$

Since the inspection should be the last task, we add a constraint $X_i + T_i \leq Inspect$ for each variable X_i, where duration T_i is the duration of task X_i.

A solution is an assignment of each variable to a value in $D_i = [0, 27]$ such that all the constraints above are satisfied.

Example of CSPs : 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. Therefore, $Axle_F$ and $Axle_B$ tasks cannot overlap in time.
How can we write this constraint ?

Example of CSPs : 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. Therefore, $Axle_F$ and $Axle_B$ tasks cannot overlap in time.
How can we write this constraint ?

Disjunctive Constraints

$$\big(Axle_F + 10 \leq Axle_B\big) \quad \textbf{or} \quad \big(Axle_B + 10 \leq Axle_F\big).$$

A linear program is an optimization problem where the objective function is linear, and the variables are subject to linear inequalities.

Example of a linear program

Minimize $Inspect$ such that

$$\forall X_i \in X - \{Inspect\} : X_i + T_i \leq Inspect;$$

$$\begin{aligned}
Axle_F + 10 &\leq Wheel_{RF}; & Axle_F + 10 &\leq Wheel_{LF};\\
Axle_B + 10 &\leq Wheel_{RB}; & Axle_B + 10 &\leq Wheel_{LB};\\
Wheel_{RF} + 1 &\leq Nuts_{RF}; & Nuts_{RF} + 2 &\leq Cap_{RF};\\
Wheel_{LF} + 1 &\leq Nuts_{LF}; & Nuts_{LF} + 2 &\leq Cap_{LF};\\
Wheel_{RB} + 1 &\leq Nuts_{RB}; & Nuts_{RB} + 2 &\leq Cap_{RB};\\
Wheel_{LB} + 1 &\leq Nuts_{LB}; & Nuts_{LB} + 2 &\leq Cap_{LB}.
\end{aligned}$$

Linear programming VS Constraint Satisfaction Problems

- Linear programming solves optimization problems.
- CSPs are not for optimization problems (we only look for a solution that satisfies all given constraints).
- Linear programming is limited to linear constraints.
- CSPs are not limited to linear constraints, they can also handle complex constraints on discrete variables (e.g., the 8-queens problem).

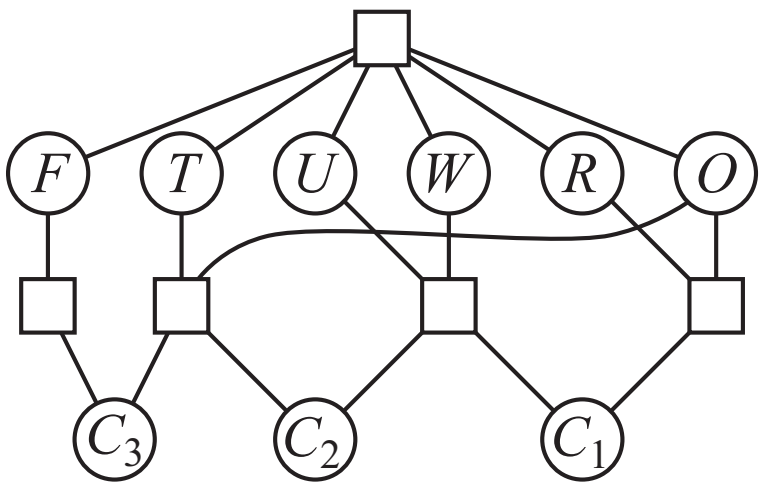
Example of CSPs : Crypt-arithmetic puzzle

- Each letter in a crypt-arithmetic puzzle represents a different digit (domain $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$).
- In this example, variables are $\{F, T, U, W, R, O, C_1, C_3, C_3\}$ where C_i are auxiliary variables representing the digit carried over.

Variables $\{F, T, U, W, R, O\}$ should be all different, we can write this as Alldiff(F,T,U,W,R,O).

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

(a)



(b)

Constraint Hypergraph

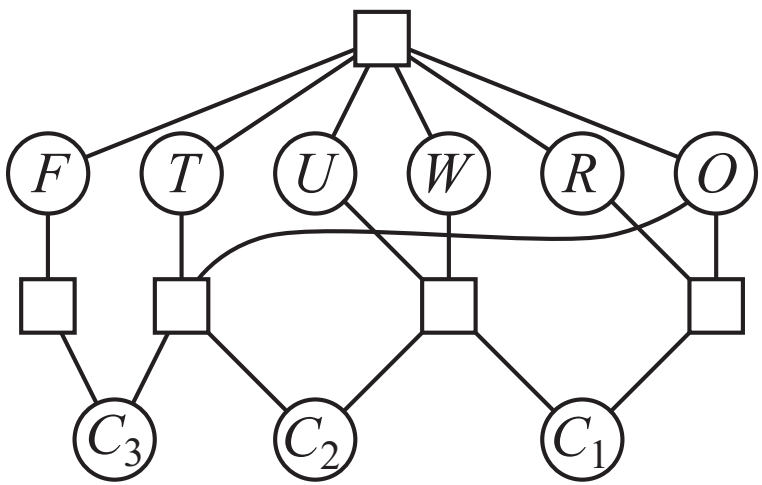
Example of CSPs : Crypt-arithmetic puzzle

The addition constraints can be written as

$$\begin{aligned} O + O &= R + 10C_1 \\ C_1 + W + W &= U + 10C_2 \\ C_2 + T + T &= O + 10C_3 \\ C_3 &= F \end{aligned}$$

$$\begin{array}{r} \quad T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

(a)



(b)

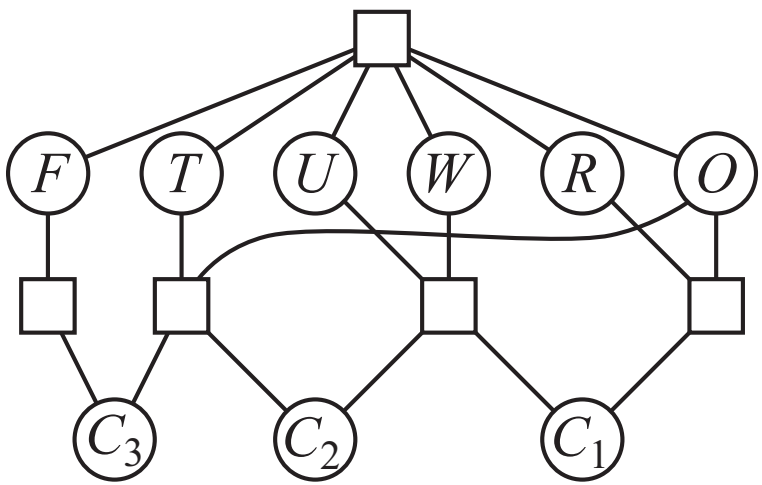
Constraint Hypergraph

Types of constraints

- Unary constraint : restricts the value of a variable. We can remove unary constraint by re-defining the domain of the corresponding variable.
- Binary constraint : a constraint on two variables (e.g., $x \neq y$), represented by a graph.
- Global constraint : a constraint on more than two variables (e.g., $x + y \neq z$), represented by a hypergraph.

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

(a)



(b)

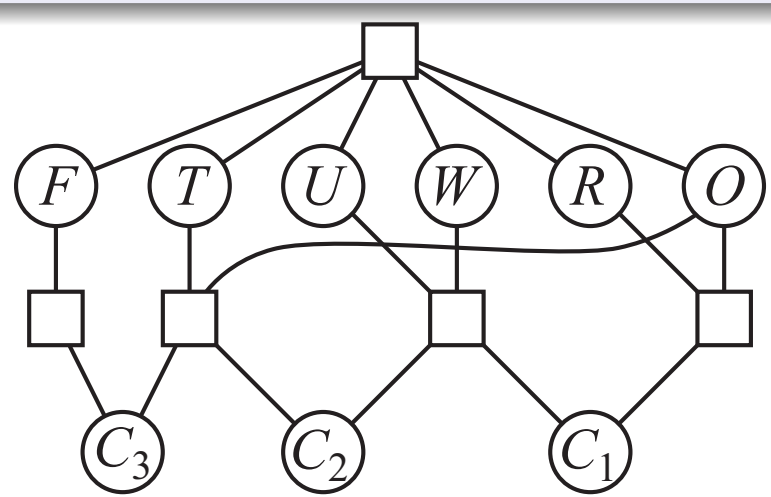
Constraint Hypergraph

Types of constraints

- Any set of global constraints can be transformed to a set of binary variables.
  Hint : transform a constraint such as $O + O = R + 10C_1$ into binary constraints
  $O + O = first(x) + 10second(x)$,
  $first(x) = R$,
  $second(x) = C_1$
- CSP solvers are generally designed for binary constraints. We assume that our constraints are binary.

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

(a)



(b)

Constraint propagation

- Constraint propagation : shrinking the domain of each variable based on the constraints.
- Results in a faster search (smaller number of values for each variable).
- Can be done as a preprocessing step before search starts, or during search.
- Works by forcing consistency : the values of a variable are reduced to a set that does not violate the constraints.
- There are several levels of consistency :
  1. Node consistency
  2. Arc consistency
  3. Path consistency
  4. K-consistency

CSP example : Sudoko

	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		

(a)

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

(b)

Node consistency

- A single variable (corresponding to a node in the CSP network) is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints.
- Example : in the previous map-coloring problem, assume the color of SA (South Australia) should not be green. Then the domain of SA is reduced from $\{green, blue, red\}$ to $\{blue, red\}$.
- A network is node-consistent if every variable in the network is node-consistent.
- Can be done in n operations, where n is the number of unary constraints.

Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints.
- X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j).
- A network is arc-consistent if every variable is arc consistent with every other variable.

Arc Consistency

Example 1

- Variables : X_1 and X_2.
- Domains : $D_1 = D_2 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (digits)
- Constraints : $X_1 = \sqrt{X_2}$
- To make X_1 arc-consistent with X_2, we need to change D_1 to $\{0, 1, 2, 3\}$.
- To make X_2 arc-consistent with X_1, we need to change D_2 to $\{0, 1, 4, 9\}$.

Example 2

- Forcing arc consistencies on the map-coloring example does not result in any modification.
- For example, the constraint $SA \neq WA$ can be satisfied by any value of SA in $\{green, blue, red\}$, and any value of WA in $\{green, blue, red\}$ as long as $SA \neq WA$.

Arc consistency algorithm (AC-3) (Mackworth, 1977)

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 $true$

---

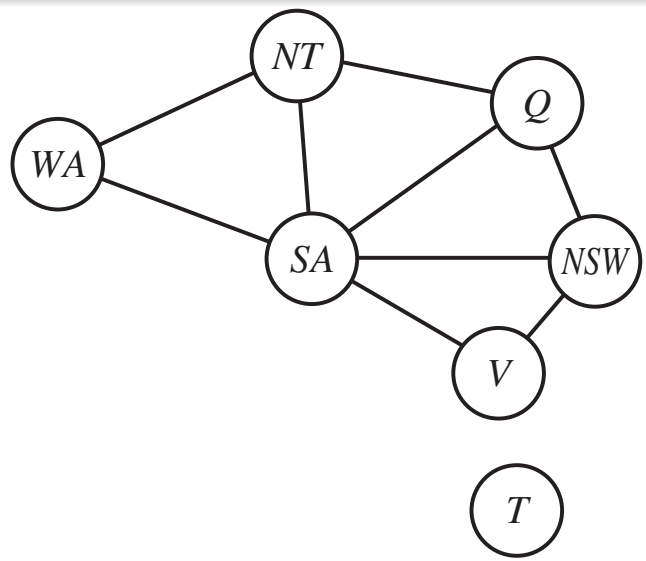
function REVISE(csp, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  $revised \leftarrow false$
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      $revised \leftarrow true$
  return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

Arc consistency algorithm (AC-3) (Mackworth, 1977)

- AC-3 algorithm eventually stops when the domains of all variables are finite.
- Complexity : $O(cd^3)$, where c is the number of constraints and d is the size of the largest domain. An arc can be inserted in the queue only d times. It takes d^2 operations to check the consistency of an arc.
- Arc consistency can be generalized to handle n-ary constraints. Replace values by tuples of values.

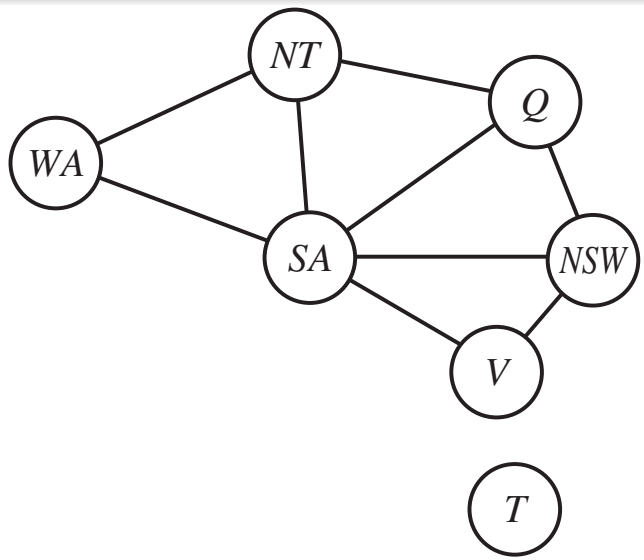
Path consistency

- Arc consistency can be useless in certain problems (map-coloring problem).
- Path consistency can do better by using implicit constraints that are inferred by looking at triples of variables.
- A two-variable set $\{X_i, X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $X_i = a, X_j = b$ consistent with the constraints on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$.



Path consistency

- Arc consistency can be useless in certain problems (map-coloring problem).
- Consider the map-coloring problem with two colors.
- A path consistency algorithm can immediately detect that there is no solution to this problem.
- The PC-2 algorithm (Mackworth, 1977) achieves path consistency in much the same way that AC-3 achieves arc consistency.

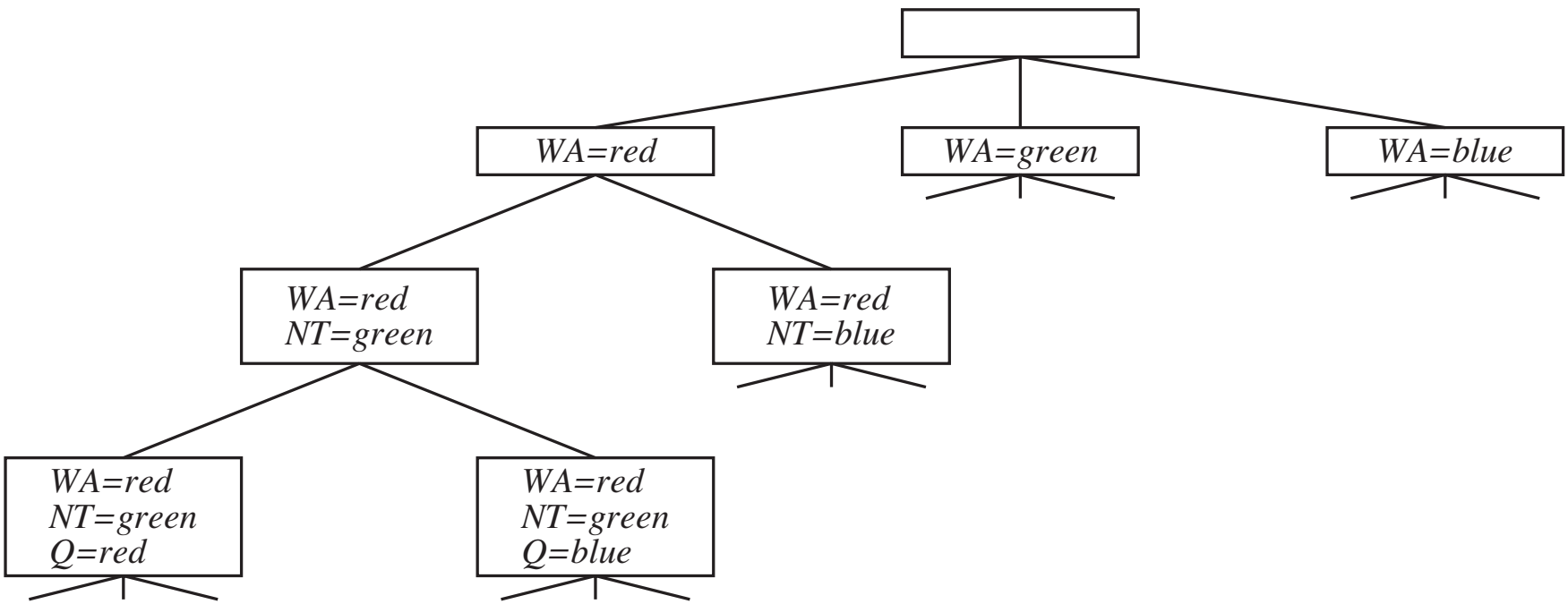


K-consistency

- 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 k^{th} variable.
- 1-consistent $=$ node-consistent, 2-consistent $=$ arc-consistent, 3-consistent $=$ path-consistent.
- A CSP is strongly k-consistent if it is k-consistent and is also $(k-1)$-consistent, $(k-2)$-consistent,. . . , 1-consistent.
- If a CSP has n variables, and it is strongly n-consistent, then a solution can be easily found by setting any variable to any value from its reduced domain.
- In general, we check the k-consistencies in the increasing values of k (from 0), we usually stop at $k=2$ or $k=3$.

Backtracking search for CSPs

- CSPs cannot be solved by inference (constraint propagation) alone.
- We can use depth-search to find a solution, but we should take into account that CSPs are commutative : the order of the assignments of values to variables does not affect the completeness of the search.
- At each level, we only consider one variable (with n possible values).
- The number of leaves is d^n, where d is the number of variables.



Backtracking search in the map-coloring problem

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** $assignment$
  $var \leftarrow$ 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 $assignment$
      $inferences \leftarrow$ INFERENCE(csp, var, $value$)
      if $inferences \neq failure$ **then**
        add $inferences$ to *csp*
        $result \leftarrow$ BACKTRACK($assignment$, csp)
        if $result \neq failure$ **then**
          return $result$
    remove $\{var = value\}$ and $inferences$ from *csp*
  return $failure$

Inferences here refers to constraint propagation (k-consistencies for a chosen k).

Which variable should be assigned next (SELECT-UNASSIGNED-VARIABLE) ?

Minimum remaining-values (MRV) heuristic

- Select the variable that is most constrained.
- Also known as "fail-first" heuristic, because it chooses a variable that can quickly fail, which results in pruning the corresponding branch of the search tree, avoiding pointless searches through other variables.
- No need for domain knowledge here.

Degree heuristic

- Reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.
- Useful at the beginning of the search, when the MRV heuristic cannot be used, or when there is a tie with MRV.

In what order should the values of a chosen variable be tried (ORDER-DOMAIN-VALUES) ?

Least-constraining-value

- Prefer the value that rules out the fewest choices for the neighboring variables in the constraint graph.
- Maximum flexibility for subsequent variable assignments.
- Intuition : Least-constraining-value is the most likely to succeed, and we want to find the first solution quickly. We will not fail before we try all the values, so there is no time economy in using a "fail-first" strategy here.