1 Constraint Satisfaction Problems

Constraint Satisfaction Problem

Constraint Satisfaction is a technique where a problem is solved when its solution satisfies certain constraints or rules of the problem.

Components:

- A state, defined by **variables** X_i with **d** values from **domain** D_i
- A goal test, defined as a set of **constraints** C specifying allowable combinations of values for subsets of variables.

Solving Constraint Satisfaction Problems:

- A state space
- Notion of the solution

Example of a Constraint Satisfaction Problem

Problem: Assign each territory a colour such that no two adjacent territories have the same colour.

Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$
Domain of Variables: $D = \{red, green, blue\}$
Constraints: $C = \{SA \neq WA, SA \neq NT, SA \neq Q, \dots\}$

1.1 Assignment of Values to Variables

A state in state-space is not a "blackbox" as in standard search, but defined by assigning values to some or all variables.

$$X_1 = v_1, X_2 = v_2, \dots, X_d = v_d$$

The assignment of these values can be done in different ways:

1. **Consistent/Legal Assignment**: An assignment is consistent if it satisfies all constraints or rules.
2. **Complete Assignment:** An assignment is complete if every variable is assigned a value, and the solution to the CSP remains consistent.
3. **Partial Assignment:** An assignment is partial if some variables are not assigned values.

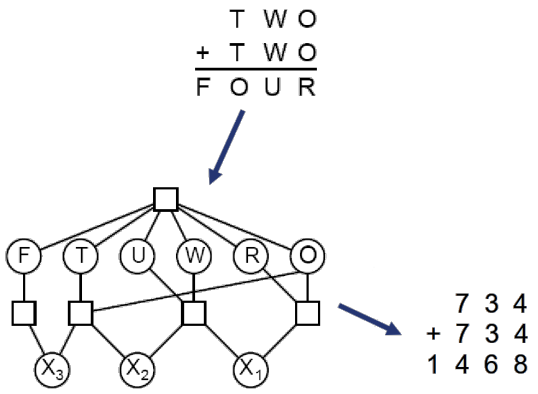
1.2 Constraint Graphs

Constraint Graphs are often constructed because abstraction of the problem makes it easier to solve and understand.

A constraint graph is usually denoted with

- Every variable is represented by a node
- Every edge indicates a constraint between two variables

Example of a Constraint Graph



Problem: Assign uniques value to the variables of each letter, so that resulting equation is true.

Variables: $X = \{T, W, O, F, U, R\}$
Domain of Variables: $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraints: $C = \{T \neq W \neq O \neq F \neq U \neq R\}$
$\cup$ $\texttt{int}(''T'' +'' W'' +'' O'') + \texttt{int}(''T'' +'' W'' +'' O'') = \texttt{int}(''F'' +'' O'' +'' U'' +'' R'')$

Here the connected nodes are involved in (in-)equations:

$$\begin{aligned} 2 \cdot O &= 10 \cdot X_1 + R \\ 2 \cdot W + X_1 &= 10 \cdot X_2 + U \\ 2 \cdot T + X_2 &= 10 \cdot X_3 + O \\ F &= X_3 \\ T \neq W \neq O &\neq F \neq U \neq R \end{aligned}$$

1.3 Types of Constraints

- **Unary constraints:** Involve a single variable (e.g. South Australia $\neq$ green)
- **Binary constraints:** Involve two variables (e.g. South Australia $\neq$ Wester Australia)
- **Higher-order constraints:** Involve more than two variables (e.g. $2 \cdot W + X_1 = 10 \cdot X_2 + U$)

Preferences / Soft constraints:

- Not binding, but should be considered during search

  $\rightarrow$ Constrained optimization problems
- e.g. Red is better than green

1.4 Solving CSPs: Search

Basic Idea:

1. Successively assign values to variable
2. Check constraints
3. If constraint is violated → backtrack
4. Repeat until all variables have assigned values that satisfy constraints

To do this we map CSPs into search problems:

- Nodes = assignments of values to a subset of the variables
- Neighbors of a node = nodes in which values are assigned to one additional variable
- Start node = empty assignment
- Goal node = a node which assigns a value to each variable and satisfies all constraints

1.4.1 Naive Search

Naive Search is practically a **brute-force** method. It systematically explores all possible assignments of v values to n variables. This, of course, is incredibly innefficient and results in exponential time complexity. The number of leaves in the search tree grows with $n!v^n$.

1.4.2 Backtracking Search

Basic Idea: As assignments are commutative ([WA = red then NT = green] = [Nt = green then WA = red]) we can reduce the number of leaves in the search tree by only considering nodes that have not been visited before.

Backtracking Algorithm

```
1  Function backtrack_search(csp):
2      return recursive_backtrack(csp, {});
3  Function recursive_backtrack(csp, assignment):
4      If is_complete(assignment) then
5          return assignment;
6      var = get_unassigned_variable(get_variables(csp), assignment, csp);
7      ForEach value in order_domain_values(var, assignment, csp) do
8          If value is consistent with assignment given constraints(csp) then
9              assignment.add({var = value});
10             result = recursive_backtrack(csp, assignment);
11             If result != failure then
12                 return result;
13             assignment.remove({var = value});
14     return failure;
```

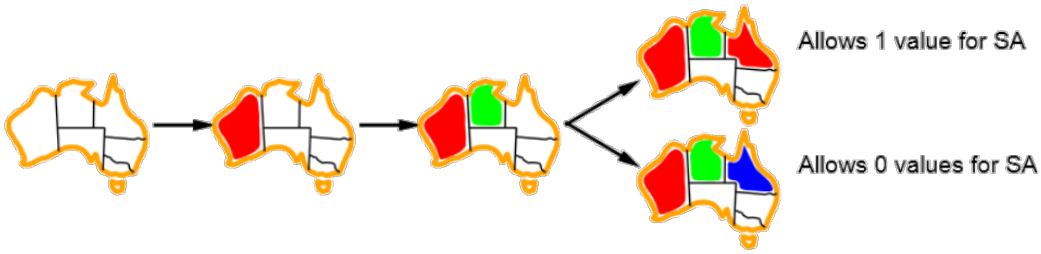
This is still not ideal as in the worst case the complexity is still exponential. This can be improved by including heuristics.

1.5 Heuristics for CSPs

- **Domain-Specific Heuristics:** Depend on the particular characteristics of the problem.
- **General-Purpose Heuristics:** Can work on any CSP.
  - **Minimum Remaining Value:** Choose variable with fewest consistent values

  - **Degree Heuristic:** Choose variable with the most constraints on remaining variables

  - **Least Constraining Value Heuristic:** Given a variable, choose the value that rules out the fewest values in the remaining variables



If utilized in this order, these heuristics will greatly improve search speed.

1.6 Constraint Propagation

Node Consistency

A variable (node) is consistent if the possible values of this variable are conform to all unary constraints.

Local Consistency

Local consistency is defined by a graph where each of its nodes is consistent with its neighbors. This is done by iteratively enforcing the constraint corresponding to the edges.

Arc

A constraint involving two variables is called an **arc** or binary constraint.

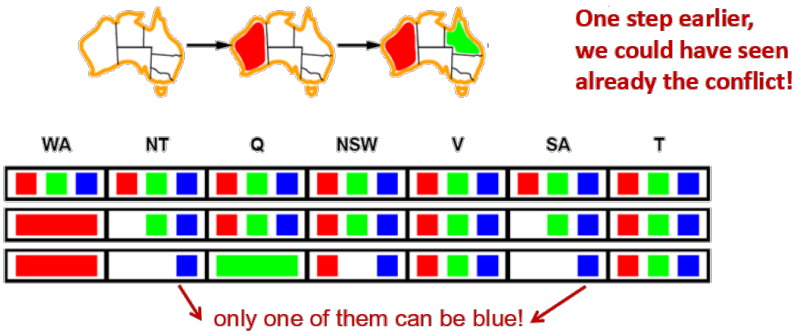
Arc Consistency

An arc is consistent if for each value of X in the domain of X there exists a value Y in the domain of Y such that the constraint $\text{arc}(X, Y)$ is satisfied.

$$\forall X \in \text{dom}(X), \exists Y \in \text{dom}(Y) : \text{arc}(X, Y) \text{ is satisfied}$$

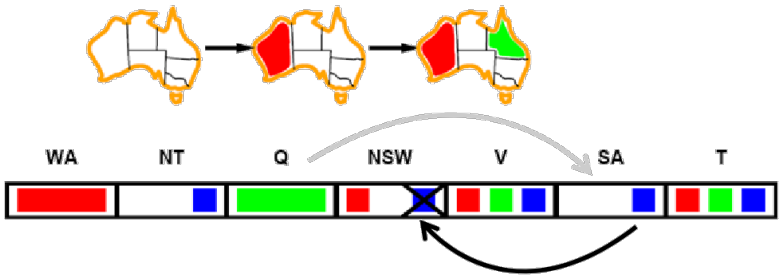
1.6.1 Forward Checking

Basic Idea: Keep track of remaining legal values for unassigned variables and terminate search, when any variable has no legal values left.

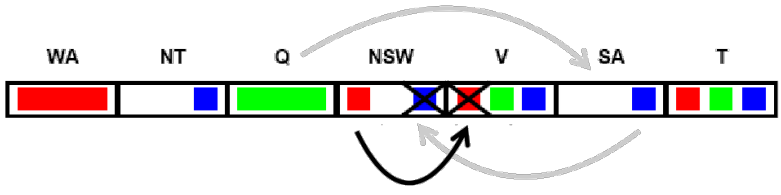


1.6.2 Maintainting Arc Consistency (MAC)

After each assignment of a value to a variable, possible values of the neighbors have to be updated.



If one variable (NSW) losses a value (blue), we need to recheck its neighbors as well because they might have lost a possible value.



AC-3 Algorithm

```
1  Function AC-3(csp):
2      queue = get_all_arcs(csp);
3      While queue is not empty do
4          (X_i, X_j) = remove_first(queue);
5          If remove_inconsistent_values(X_i, X_j) then
6              ForEach X_k in get_neighbors(X_i) do
7                  queue.add(X_k);
8  Function remove_inconsistent_values(X_i, X_j):
9      removed = false;
10     ForEach x in get_domain(X_i) do
11         If no value y in get_domain(X_j) satisfies arc(X_i, X_j) then
12             get_domain(X_i).remove(x);
13             removed = true;
14     return removed;
```

1.6.3 Path Consistency

Arc consistency is often sufficient to:

- Solve the problem (all variable domains reduced to one value)
- Show that the problem cannot be solved (some domains empty)

but sometimes may not be enough, for example if theres always a consistent value in the neighboring region.

Path consistency tightens the binary constraint by considering triples of values.

A pair of variables (X_i, X_j) is path-consistent with X_m if

- for every assignment that satisfies the constraint on the arc (X_i, X_j)
- there is an assignment that satisfies the constraints on the arcs (X_i, X_m) and (X_j, X_m)

1.6.4 k-Consistency

k-Consistency is a generalization of path consistency. A set of k values need to be consistent. It may lead to a faster solution but checking for k-consistency is computationally expensive with exponential time in the worst case.

In practice, arc consistency is most frequently used.

1.6.5 Constrain Propagation & Backtracking Search

Basic Idea: Each time a variable is assigned, a constraint propagation algorithm is run in order to reduce the number of choice points in the search. This can improve the speed of backtracking search further. This algorithm can be implemented using Forward Checking or AC-3.

1.7 Local Search for CSPs

Neccessary Modifications for CSPs:

- work with complete states
- allow states with unsatisfied constraints
- operators reassign variable values

Min-Conflicts Heuristic:

- Randomly select a conflicted variable
- Choose the value that violates fewest constraints
- Hill climbing with h(n) = # of violated constraints

Performance:

- Can solve randomly generated CSPs with a high probability
- Except in a narrow range $R = \frac{\text{\# of constraints}}{\text{\# of variables}}$

1.8 Problem Decomposition

Assume search space for a constraint satisfaction with **n variables**, each of which can have **d values** = $O(d^n)$

Basic Idea: Decompose the problem into subproblems with **c variables** each

- Each problem has complexity $O(d^c)$
- There are n/c problems → total complexity $O(n/c \cdot d^c)$
- Unconditional independence is rare

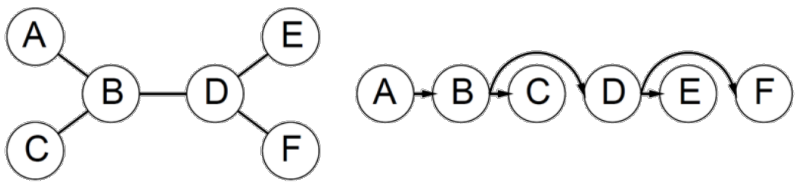
This can reduce the total complexity from exponential to linear, assuming c is constant.

1.9 Tree-Structured CSPs

A CSP is tree-structured if in the constraint graph any two variables are connected by a single path. Any tree structured CSP can be solved in linear time in the number of variables = $O(n \cdot d^2)$

1.9.1 Linear Algorithm

1. Choose variable as root, order nodes so that parent always comes before its children (only one parent per node)
2. For j = n downto 2
   - Make the arc(X_i, X_j) arc-consistent, calling `remove_inconsistent_value(`X_i, X_j`)`
3. For i = 1 to n
   - Assign to X_i any value that is consistent with its parent



1.9.2 Nearly Tree-Structured Problems

Tree structured problems are rare.

Approaches for making them tree-structured:

1. **Cutset Conditioning:**
   - Removing nodes so that the remaining nodes form a tree
2. **Collapsing nodes together:**
   - Decompose the graph into a set of independent tree-shaped subproblems

Cutset Conditioning

1. Choose a subset S of the variables such that the constraint graph becomes a tree after removal of S (cycle cutset)
2. Choose a consistent assignment of variables for S
3. Remove from the remaining variables all values that are inconsistent with the variables of S
4. Solve the CSP problem with the remainign variables
5. If no solution: Choose different S in **2**