Constraint Satisfaction Problems

- A Constraint Satisfaction Problems (CSP) consists of
  - a set of variables \( \{X_1, X_2, \ldots, X_n\} \) to which
  - values \( \{d_1, d_2, \ldots, d_k\} \) can be assigned
  - such that a set of constraints over the variables is respected

- A CSP is solved by a variable assignment that satisfies all given constraints.

In CSPs, states are explicitly represented as variable assignments. CSP search algorithms take advantage of this structure.

- The main idea is to exploit the constraints to eliminate large portions of search space.

- Formal representation language with associated general inference algorithms

Example: Map-Coloring

- Variables: \( WA, NT, SA, Q, NSW, V, T \)
- Values: \{red, green, blue\}
- Constraints: adjacent regions must have different colors, e.g., \( NSW \neq V \)
Australian Capital Territory (ACT) and Canberra (inside NSW)

View of the Australian National University and Telstra Tower

- **Solution assignment:**
  - \{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}
  - Perhaps in addition \(ACT = blue\)

- **Constraint Graph**
  - A constraint graph can be used to visualize binary constraints
  - For higher order constraints, hyper-graph representations might be used
  - Nodes = variables, arcs = constraints

  Note: Our problem is 3-colorability for a planar graph

- **Variations**
  - Binary, ternary, or even higher arity (e.g., ALL_DIFFERENT)
  - Finite domains (d values) \(\rightarrow d^n\) possible variable assignments
  - Infinite domains (reals, integers)
    - Linear constraints: solvable (in P if real)
    - Nonlinear constraints: unsolvable
Applications

- Timetabling (classes, rooms, times)
- Configuration (hardware, cars, ...)  
- Spreadsheets
- Scheduling
- Floor planning
- Frequency assignments
- Sudoku
- ...

Algorithm

```plaintext
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 ← INERENCE(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 over Assignments

- Assign values to variables step by step (order does not matter)
- Consider only one variable per search node!
- DFS with single-variable assignments is called backtracking search
- Can solve n-queens for \( n \approx 25 \)

Example (1)
Improving Efficiency:
CSP Heuristics & Pruning Techniques

- **Variable ordering**: Which one to assign first?
- **Value ordering**: Which value to try first?
- **Try to detect failures** early on
- **Try to exploit problem structure**

→ **Note**: all this is not problem-specific!
Variable Ordering:
Most constrained first

- Most constrained variable:
  - choose the variable with the fewest remaining legal values
  → reduces branching factor!

Value Ordering:
Least Constraining Value First

- Given a variable,
  - choose first a value that rules out the fewest values in the remaining unassigned variables
  → We want to find an assignment that satisfies the constraints (of course, does not help if unsat.)

Rule out Failures early on:
Forward Checking

- Whenever a value is assigned to a variable, values that are now illegal for other variables are removed
- Implements what the ordering heuristics implicitly compute
- WA = red, then NT cannot become red
- If all values are removed for one variable, we can stop!
Forward Checking (1)

- Keep track of remaining values
- Stop if all have been removed

Forward Checking (2)

- Keep track of remaining values
- Stop if all have been removed

Forward Checking (3)

- Keep track of remaining values
- Stop if all have been removed

Forward Checking (4)

- Keep track of remaining values
- Stop if all have been removed
Forward Checking:
Sometimes it Misses Something

- Forward Checking propagates information from assigned to unassigned variables
- However, there is no propagation between unassigned variables

Arc Consistency

- A directed arc $X \rightarrow Y$ is “consistent” iff for every value $x$ of $X$, there exists a value $y$ of $Y$, such that $(x, y)$ satisfies the constraint between $X$ and $Y$
- Remove values from the domain of $X$ to enforce arc-consistency
- Arc consistency detects failures earlier
- Can be used as preprocessing technique or as a propagation step during backtracking

Arc Consistency Example

AC3 Algorithm

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

while queue is not empty do
    $(X_i, X_j) \leftarrow$ REMOVED-FIRST(queue)
    if REVISE(esp, $X_i$, $X_j$) 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(esp, $X_i$, $X_j$) 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
```
Properties of AC3

- AC3 runs in $O(d^3 n^2)$ time, with $n$ being the number of nodes and $d$ being the maximal number of elements in a domain
- Of course, AC3 does not detect all inconsistencies (which is an NP-hard problem)

Problem Structure (1)

- CSP has two independent components
- Identifiable as connected components of constraint graph
- Can reduce the search space dramatically

Problem Structure (2):
Tree-structured CSPs

- If the CSP graph is a tree, then it can be solved in $O(nd^2)$
- General CSPs need in the worst case $O(d^n)$
- Idea: Pick root, order nodes, apply arc consistency from leaves to root, and assign values starting at root

Problem Structure (2):
Tree-structured CSPs

- Pick any variable as root; choose an ordering such that each variable appears after its parent in the tree.
- Apply arc-consistency to $(X_i, X_k)$, when $X_i$ is the parent of $X_k$, for all $k = n$ down to 2. (any tree with $n$ nodes has $n-1$ arcs, per arc $d^2$ comparisons are needed: $O(n^2 d^2)$)
- Now one can start at $X_1$ assigning values from the remaining domains without creating any conflict in one sweep through the tree!
- Algorithm linear in $n$
Problem Structure (3):
Almost Tree-structured

Idea: Reduce the graph structure to a tree by fixing values in a reasonably chosen subset

![Graph transformation diagram]

Instantiate a variable and prune values in neighboring variables is called Conditioning

Problem Structure (4):
Almost Tree-structured

Algorithm Cutset Conditioning:
1. Choose a subset S of the CSPs variables such that the constraint graph becomes a tree after removal of S. S is called a cycle cutset.
2. For each possible assignment of variables in S that satisfies all constraints on S
   - remove from the domains of the remaining variables any values that are inconsistent with the assignments for S, and
   - if the remaining CSP has a solution, return it together with the assignment for S

Note: Finding the smallest cycle cutset is NP hard, but several efficient approximation algorithms are known.

Another Method:
Tree Decomposition (1)

- Decompose problem into a set of connected sub-problems, where two sub-problems are connected when they share a constraint
- Solve sub-problems independently and combine solutions

Another Method:
Tree Decomposition (2)

- A tree decomposition must satisfy the following conditions:
  - Every variable of the original problem appears in at least one sub-problem
  - Every constraint appears in at least one sub-problem
  - If a variable appears in two sub-problems, it must appear in all sub-problems on the path between the two sub-problems
  - The connections form a tree
Another Method: Tree Decomposition (3)

- Consider sub-problems as new mega-variables, which have values defined by the solutions to the sub-problems.
- Use technique for tree-structured CSP to find an overall solution (constraint is to have identical values for the same variable).

Tree Width

- The aim is to make the subproblems as small as possible. Tree width $w$ of a tree decomposition is the size of largest sub-problem minus 1.
- Tree width of a graph is minimal tree width over all possible tree decompositions.
- If a graph has tree width $w$ and we know a tree decomposition with that width, we can solve the problem in $O(nd^{w+1})$.
- Unfortunately, finding a tree decomposition with minimal tree width is NP-hard. However, there are heuristic methods that work well in practice.

Summary & Outlook

- CSPs are a special kind of search problem:
  - states are value assignments
  - goal test is defined by constraints
- Backtracking = DFS with one variable assigned per node. Other intelligent backtracking techniques possible.
- Variable/value ordering heuristics can help dramatically.
- Constraint propagation prunes the search space.
- Path-consistency is a constraint propagation technique for triples of variables.
- Tree structure of CSP graph simplifies problem significantly.
- Cutset conditioning and tree decomposition are two ways to transform part of the problem into a tree.
- CSPs can also be solved using local search.