Constraint Satisfaction Problems

AIMA-3rd Chapter 6

Outline

- ♢ CSP examples
- ♢ Backtracking search for CSPs
- ♢ Problem structure and problem decomposition
- ♢ Local search for CSPs

Constraint satisfaction problems (CSPs)

Standard search problem:

state is a "black box"– any old data structure that supports goal test, eval, successor

CSP:

- ♢ state is defined by **variables** X_i with **values** from **domain** D_i: **factor representation**
- ♢ goal test is a set of **constraints** specifying allowable combinations of values for subsets of variables
- ♢ A problem is solved when each variable has a value that satisfies all the constraints.

CSP allows useful *general-purpose* heuristic algorithms with more power than standard search algorithms that use *problem-specific* heuristics by taking advantage of the structure of states.

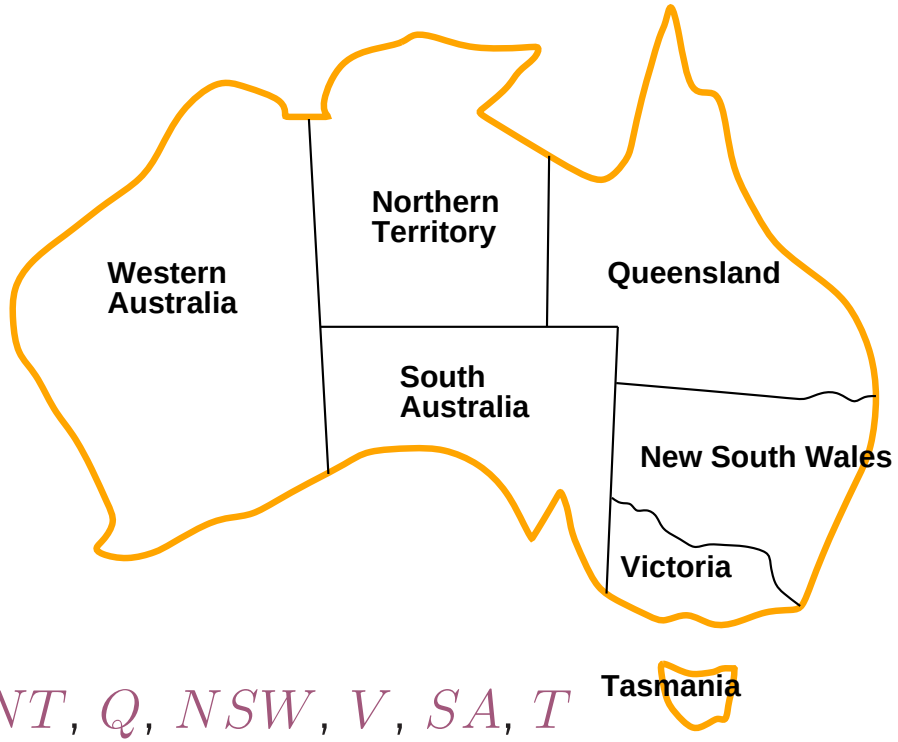
Defining CSPs

CSP consists of three components, X, D, and C:
– set of variables $X = X_1, X_2, ..., X_n$.
– set of domains $D = D_1, D_2, ..., D_n$.
– set of constraints that specify allowable combinations of values.

D_i consists of a set of allowable values, $v_1, ..., v_k$ for X_i.
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.

ex. "two variables must have different values":
$< (X_1, X_2), [(A, B), (B, A)] >$ or $< (X_1, X_2), X_1 != X_2 >$.

Example: Map-Coloring



Variables WA, NT, Q, NSW, V, SA, T

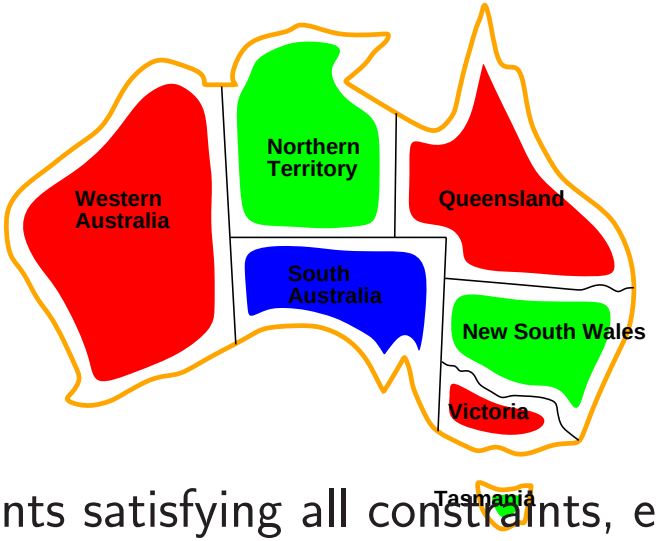
Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g., $WA \neq NT$ (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring contd.



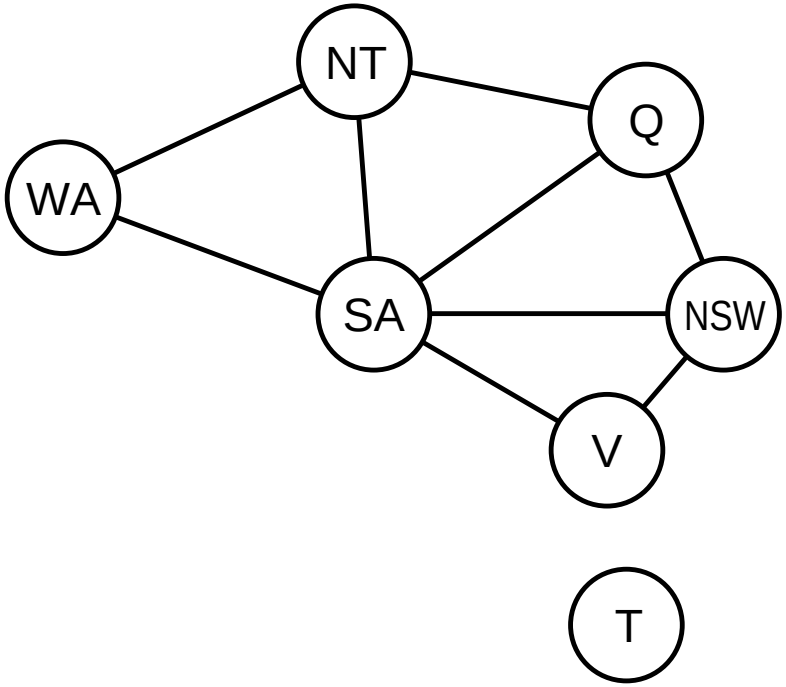
Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Solution to CSP is **consistent, complete assignment** of variables that defines the states.
Consistent assignment assigns values to variables such that all asignment is legal.
Complete assignment assigns values to all variable in the state.

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Types of Vairable in CSP

Discrete variables

Finite domains; size $d \Rightarrow O(d^n)$ complete assignments

– e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)

– e.g., 8-queens problem where variables Q_i are the positions of each queen in columes 1...8.

Infinite domains (integers, strings, etc.)

– e.g., job scheduling without a deadline, variables are start/end days for each job

– need a **constraint language**, e.g., precedence constraint $StartJob_1 + 5 \leq StartJob_3$

– linear constraints on integers solvable, but no solutions to nonlinear constrains on integer variable.

Continuous variables

– e.g., start/end times for Hubble Space Telescope observations

– linear constraints solvable in polynomial time by Linear Programming methods, where constraints must be linear equalities or inequalities.

Types of constraints in CSP

Unary constraints involve a single variable,
e.g., $SA \neq green$

Binary constraints involve pairs of variables,
e.g., $SA \neq WA$

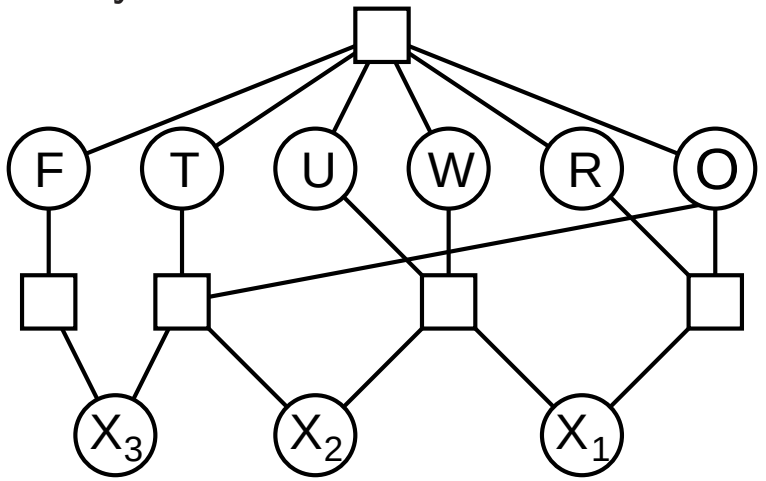
Global constraints involve arbitrary number of variables,
e.g., cryptarithmetic column constraints

Preferences (soft) constraints, e.g., red is better than $green$
often representable by a cost for each variable assignment
$\rightarrow$ constrained optimization problems

Example: Cryptarithmetic puzzles

Example of global constraint and n-ary constraints.

$$\begin{array}{cccc} & \mathsf{T} & \mathsf{W} & \mathsf{O} \\ + & \mathsf{T} & \mathsf{W} & \mathsf{O} \\ \hline \mathsf{F} & \mathsf{O} & \mathsf{U} & \mathsf{R} \end{array}$$



Variables: $F\ T\ U\ W\ R\ O$ and aux. variable $X_1\ X_2\ X_3$

Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints

$alldiff(F, T, U, W, R, O)$

$O + O = R + 10 \cdot X_1$

$X_1 + W + W = U + 10 \cdot X_2$

$X_2 + T + T = O + 10 \cdot X_3$

$X_3 = F$.

Real-world CSPs

Real-world problems are mix of hard constaints and perference constrains.

Assignment problems
    e.g., who teaches what class
Timetabling problems
    e.g., which class is offered when and where?
Hardware configuration
Spreadsheets
Transportation scheduling
Factory scheduling
Floorplanning

* Notice that many real-world problems involve real-valued variables

Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

♢ Initial state: the empty assignment, $\{\}$

♢ Successor function: assign a value to an unassigned variable
that does not conflict with current assignment.
$\Rightarrow$ fail if no legal assignments (not fixable!)

♢ Goal test: the current assignment is complete

– This is the same for all CSPs! ☺
– Every solution appears at depth n with n variables
$\Rightarrow$ use depth-first search
– Path is irrelevant, so can also use complete-state formulation
– $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!! ☹

Constraint propagation: Local consistancies

In CSP, an algorithm can not only search but also choose to do a specific type of **inference** called **constaint propagation**.

Constaint propagation:

– use the constraints to reduce the number of legal values for a variable, which propagates to reduce legal values for another variable.

– can be intertwined with search, or maybe done as prepocessing step.

The process of enforcing **local consistency** in each part of the constaint graph causes inconsistent values to be eliminated thought the graph.

– Node consistency
– Arc consistency
– Path consistency
– K-consistency
– Global-consistency

Node consistency

A single variable in the constraint graph is **node-consistent** if all the values in the variable's domain satisfy the variable's unary constraints.

A network (graph) is node-consistent if every variable in the network is node-consistent.

All unary constaints can be transformed into binary one. (we will look at binary constaint solvers and assume unary constaints can be solved as well.)

Arc-consistency

A variable CSP is **arc-consistent** if every value in its domain satisfies the variable's binary constraints

A network (graph) is arc-consistent if every variable in the network is arc-consistent.

E.g., constaint $Y = X^2$ where domain of X and Y are digits
explicit form: $< (X, Y), (0, 0), (1, 1), (2, 4), (3, 9) >$

Arc-consistency: algorithm AC-3

```
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) ← 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 ← 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 ← true
  return revised
```

Arc-consistency: algorithm AC-3 cont.

Queue maintained can be just a set. (order does not matter)

Complexity of AC-3: $O(cd^3)$

Assume a CSP with n variables, each with domain size at most d, and with c binary constraints (arcs).

Each arc (X_k, X_i) can be inserted in the queue only d times because X_i has at most d values to delete.

Checking consistency of an arc can be done in $O(d^2)$ time,

so we get $O(cd^3)$ total worst-case time.

Arc consistency can be generalized to n-ary constaints. (**generalized arc consistent**)

Path-consistency

Path consistency tightens the binary constraints 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 satises the constraints on X_i, X_m and X_m, X_j.

* 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 $k1$ variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.

1-consistency says that, given the empty set, we can make any set of one variable consistent (node consistency)
2-consistency is the same as arc consistency.
For binary constraint networks, 3-consistency is the same as path consistency.

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

Global-consistency

Note: global constraint is one involving an arbitrary number of variables (but not necessarily all variables)

Global constraints occur frequently in real problems and can be handled by special-purpose algorithms that are more efcient than the general-purpose methods described so far. e.g., $Alldiff$

One simple form of inconsistency detection for Alldiff constraints works as follows:

if m variables are involved in the constraint, and if they have n possible distinct values altogether, and $m > n$, then the constraint cannot be satised.

Generalized algorithm for detecting inconsistency in $Alldiff$:

– remove any variable in the constraint that has a singleton domain,
– delete that variables value from the domains of the remaining variables
– repeat as long as there are singleton variables.
– If at any point an empty domain is produced or there are more variables than domain values left, then an inconsistency has been detected.

Global-consistency

resource constraint ($atmost$ constraint)

e.g., The constraint that no more than 10 personnel are assigned in total is written as $Atmost(10, P_1, P_2, P_3, P_4)$.

CSP is **bounds consistent** if for every variable X, and for both the lower-bound and upper-bound values of X, there exists some value of Y that satises the constraint between X and Y for every variable Y

Backtracking search

Some problems,e.g. Sudoku can be solved by inference over constraints. But many other CSPs cannot be solved by inference alone; we must search for a solution as well.

We utilized commutative property of CSPs. A proble is commutative if the order of application of any given set of action has no effect on the outcome.

$\longrightarrow$ Only need to consider assignments to a $single$ variable at each node

$\Rightarrow \quad b = d$ and there are d^n leaves

$O(d^n)$ leaves constructed.

Depth-first search for CSPs with single-variable assignments and backtracks when a variable has no legal values left to assign is called **backtracking search**

– basic uninformed algorithm for CSPs

– keeps only a single representation of a state and alters that representation rather than creating new ones

Backtracking search algorithm

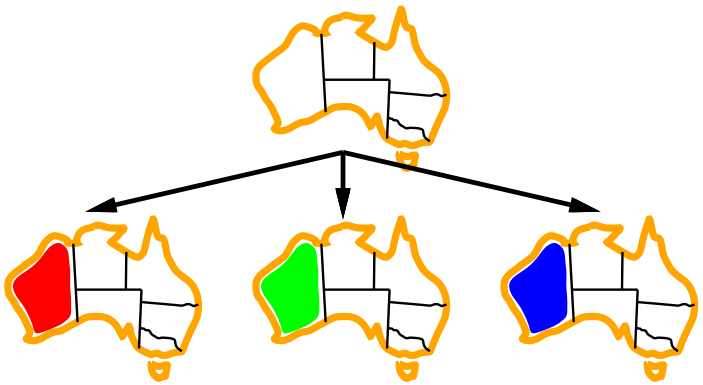
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

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
```

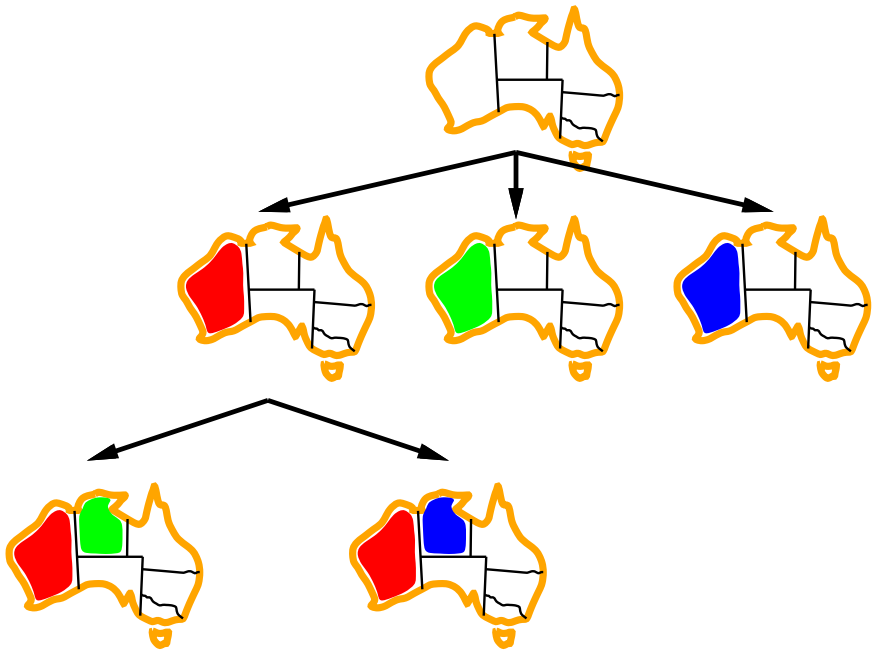
Figure 6.5 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter **??**. By varying the functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, we can implement the general-purpose heuristics discussed in the text. The function INFERENCE can optionally be used to impose arc-, path-, or *k*-consistency, as desired. If a value choice leads to failure (noticed either by INFERENCE or by BACKTRACK), then value assignments (including those made by INFERENCE) are removed from the current assignment and a new value is tried.

Backtracking example

Backtracking example



Backtracking example



Backtracking example