Constraint Satisfaction Problems (CSPs)

Russell and Norvig Chapter 5

CSP example: map coloring

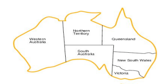
Given a map of Australia, color it using three colors such that no neighboring territories have the same color.

CSP example: map coloring

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

Constraint satisfaction problems

- A CSP is composed of:
  - A set of variables $X_1,X_2,\ldots,X_n$ with domains (possible values) $D_1,D_2,\ldots,D_n$
  - A set of constraints $C_1,C_2, \ldots,C_m$
  - Each constraint C_i limits the values that a subset of variables can take, e.g., $V_1 \neq V_2$



In our example:

- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: D_i*={red,green,blue}*
- Constraints:adjacent regions must have different colors.
  - E.g. *WA ≠ NT* (if the language allows this) or
  - (WA,NT) in {(red,green),(red,blue),(green,red),(green,blue),(blue,red),(blue,green)}

Constraint satisfaction problems

- A state is defined by an assignment of values to some or all variables.
- Consistent assignment: assignment that does not not violate the constraints.
- Complete assignment: every variable is mentioned.
- Goal: a complete, legal assignment.

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

Constraint satisfaction problems

- Simple example of a formal representation language
- CSP benefits
  - Standard representation language
  - Generic goal and successor functions
  - Useful general-purpose algorithms with more power than standard search algorithms, including generic heuristics
- Applications:
  - Time table problems (exam/teaching schedules)
  - Assignment problems (who teaches what)

Varieties of CSPs

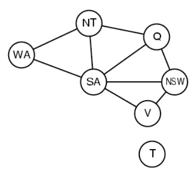
- Discrete variables
  - Finite domains of size $d \Rightarrow O(d^n)$ complete assignments.
    - The satisfiability problem: a Boolean CSP
  - Infinite domains (integers, strings, etc.)
    - E.g. job scheduling where variables are start/end days for each job.
    - Need a constraint language e.g $StartJob_1 + 5 \leq StartJob_2$.
- Continuous variables
  - e.g. start/end times for Hubble Telescope observations.
  - Linear constraints solvable in poly time by linear programming methods (dealt with in the field of operations research).

Varieties of constraints

- Unary constraints involve a single variable.
  - e.g. $SA \neq green$
- Binary constraints involve pairs of variables.
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables.
- Preference (soft constraints) e.g. *red* is better than *green* often representable by a cost for each variable assignment; not considered here.

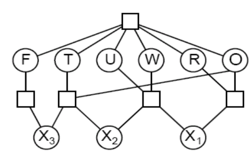
Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, edges are constraints



Example: cryptharithmetic puzzles

```
    T W O
  + T W O
  F O U R
```



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
Constraints
  alldiff(F,T,U,W,R,O)
  $O + O = R + 10 \cdot X_1$, etc.

CSP as a standard search problem

- Incremental formulation
  - *Initial State*: the empty assignment {}.
  - *Successor function*: Assign value to unassigned variable provided that there is not conflict.
  - *Goal test*: the current assignment is complete.
- Same formulation for all CSPs !!!
- Solution is found at depth *n* (*n* variables).
  - What search method would you choose?

Backtracking search

- Observation: the order of assignment doesn't matter
  $\Rightarrow$ can consider assignment of a single variable at a time. Results in d^n leaves.
- Backtracking search: DFS for CSPs with single-variable assignments (backtracks when a variable has no value that can be assigned)
- The basic uninformed algorithm for CSP

Backtracking search

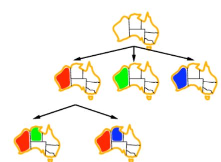
```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or
                                                        failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to
                    CONSTRAINTS[csp] then
                add {var=value} to assignment
                result ← RECURSIVE-BACTRACKING(assignment, csp)
                if result ≠ failure then return result
                remove {var=value} from assignment
    return failure
```

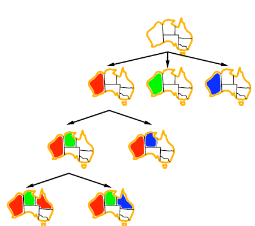

Backtracking example

Backtracking example

Backtracking example



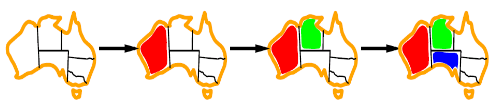
Backtracking example



Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

Most constrained variable

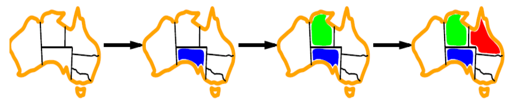


var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)

Choose the variable with the fewest legal values
(most constrained variable)
a.k.a minimum remaining values (MRV) or “fail first” heuristic

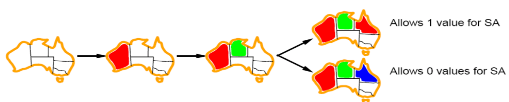
- What is the intuition behind this choice?

Most constraining variable



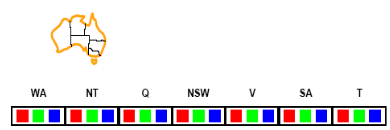
- Select the variable that is involved in the largest number of constraints on other unassigned variables.
- Also called the *degree* heuristic because that variable has the largest degree in the constraint graph.
- Often used as a tie breaker e.g. in conjunction with MRV.

Least constraining value



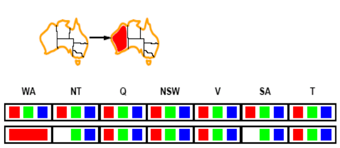
- Least constraining value heuristic: guides the choice of which value to assign next.
- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables
  - why?

Forward checking



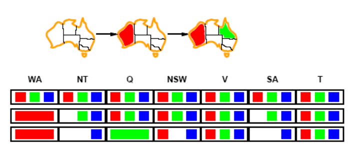
- Can we detect inevitable failure early?
  - *And avoid it later?*
- *Forward checking:* keep track of remaining legal values for unassigned variables.
- Terminate search direction when a variable has no legal values.

Forward checking



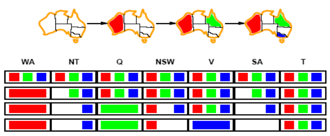
- Assign *{WA=red}*
- Effects on other variables connected by constraints with WA
  - *NT can no longer be red*
  - *SA can no longer be red*

Forward checking



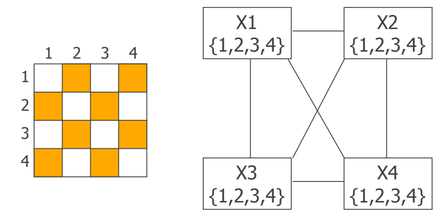
- Assign *{Q=green}*
- Effects on other variables connected by constraints with WA
  - *NT can no longer be green*
  - *NSW can no longer be green*
  - *SA can no longer be green*

Forward checking

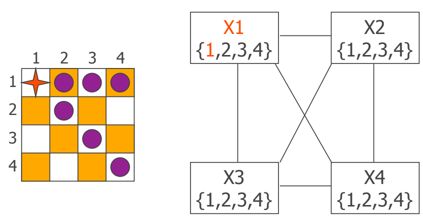


- If *V* is assigned *blue*
- Effects on other variables connected by constraints with WA
  - *SA is empty*
  - *NSW can no longer be blue*
- FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.

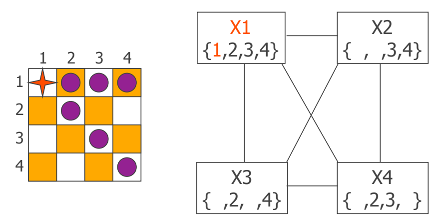
Example: 4-Queens Problem



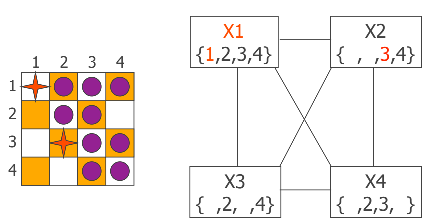
Example: 4-Queens Problem



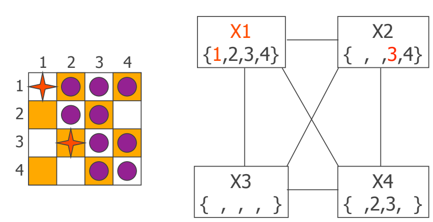
Example: 4-Queens Problem



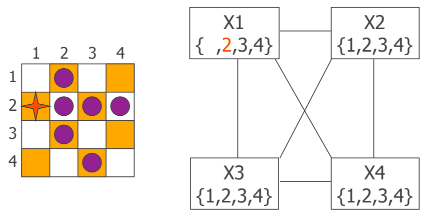
Example: 4-Queens Problem



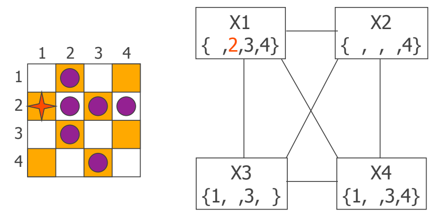
Example: 4-Queens Problem



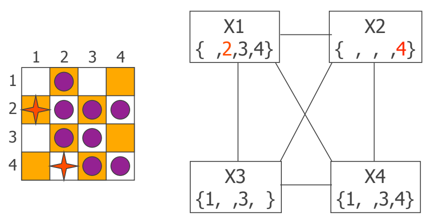
Example: 4-Queens Problem



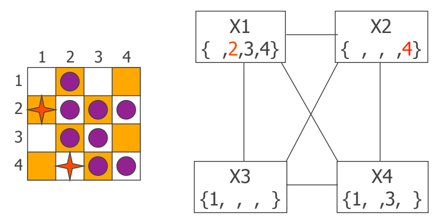
Example: 4-Queens Problem



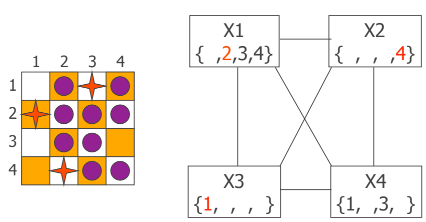
Example: 4-Queens Problem



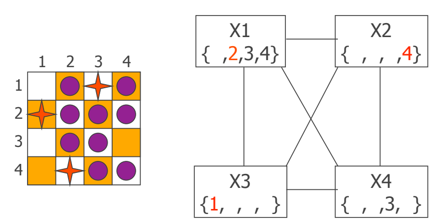
Example: 4-Queens Problem



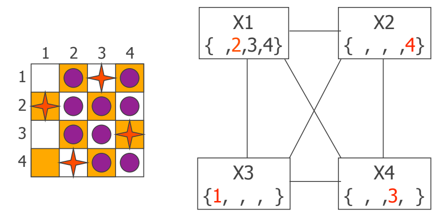
Example: 4-Queens Problem



Example: 4-Queens Problem



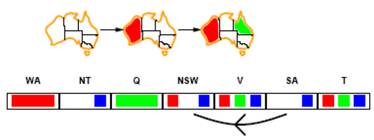
Example: 4-Queens Problem



Constraint propagation

- Solving CSPs with combination of heuristics plus forward checking is more efficient than either approach alone.
- FC does not provide early detection of all failures.
  - Once WA=red and Q=green: NT and SA cannot both be blue!
- Constraint propagation: propagate the implications of each constraint

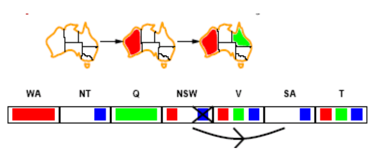
Arc consistency



- *X* → *Y* is consistent iff

  for *every* value *x* of *X* there is some allowed *y*
- *SA* → *NSW* is consistent since *SA=blue* and *NSW=red* is a consistent assignment.
- Arc – directed edge

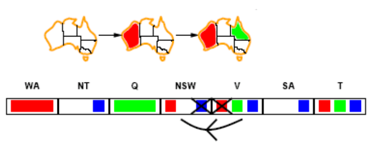
Arc consistency



- *X* → *Y* is consistent iff

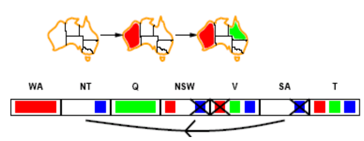
  for *every* value *x* of *X* there is some allowed *y*
- *NSW* → *SA* is not consistent since for *NSW=blue* there is no consistent assignment to *SA*.
- Arc can be made consistent by removing *blue* from *NSW*

Arc consistency



- RECHECK neighbours !!
  - Remove red from *V*

Arc consistency



- Can be run as a preprocessing before the search or after each assignment.
  - Repeated until no inconsistency remains

Arc Consistency Algorithm

function AC-3(*csp*) **return** the CSP, possibly with reduced domains
inputs: *csp*, a binary csp with variables $\{X_1, X_2, \ldots, X_n\}$
local variables: *queue*, a queue of arcs initially the arcs in *csp*

while queue is not empty **do**
$(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
for each X_k **in** NEIGHBORS[X_i] **do**
add (X_i, X_j) to queue

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **return** *true* iff we remove a value
removed $\leftarrow$ *false*
for each *x* **in** DOMAIN[X_i] **do**
if no value *y* in DOMAIN[X_j] allows (x,y) to satisfy the constraints between X_i and X_j
then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
return *removed*

Time complexity: $O(n^2d^3)$

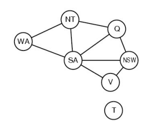
K-consistency

- Arc consistency does not detect all inconsistencies:
  - Partial assignment *{WA=red, NSW=red}* is inconsistent.
- Stronger forms of propagation can be defined using the notion of 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.
- Not practical!

Local search for CSP

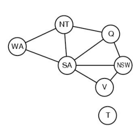
- Local search methods use a "complete" state representation, i.e., all variables assigned.
- To apply to CSPs
  - Allow states with unsatisfied constraints
  - operators reassign variable values
- Select a variable: random conflicted variable
- Select a value: *min-conflicts heuristic*
  - Value that violates the fewest constraints
  - Hill-climbing like algorithm with the objective function being the number of violated constraints
- Works surprisingly well in problem like n-Queens

Problem structure



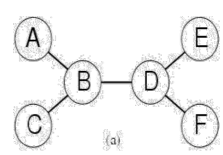
- *How can the problem structure help to find a solution quickly?*
- Subproblem identification is important:
  - Coloring Tasmania and mainland are independent subproblems
  - Identifiable as connected components of constraint graph.
- Improves performance

Problem structure



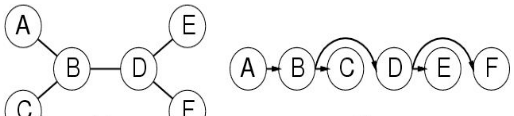
- Suppose each problem has *c* variables out of a total of *n*.
- Worst case solution cost is $O(n/c\ d^c)$ instead of $O(d^n)$
- Suppose *n=80, c=20, d=2*
  - 2^{80} = 4 billion years at 1 million nodes/sec.
  - 4 * 2^{20}= .4 second at 1 million nodes/sec

Tree-structured CSPs



- Theorem: if the constraint graph has no loops then CSP can be solved in $O(nd^2)$ time
- Compare with general CSP, where worst case is $O(d^n)$

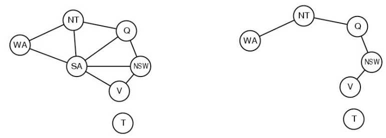
Tree-structured CSPs



(a) (b)

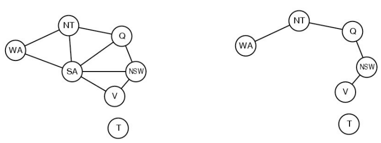
- Any tree-structured CSP can be solved in time linear in the number of variables.
  - Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering. (label var from X_1 to X_n)
  - For *j* from *n* down to 2, apply REMOVE-INCONSISTENT-VALUES(Parent(X_j), X_j)
  - For *j* from 1 to *n* assign X_j consistently with Parent(X_j)

Nearly tree-structured CSPs



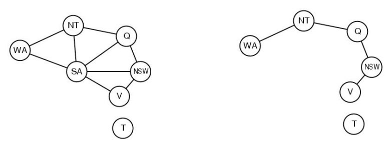
- *Can more general constraint graphs be reduced to trees?*
- Two approaches:
  - Remove certain nodes
  - Collapse certain nodes

Nearly tree-structured CSPs



- Idea: assign values to some variables so that the remaining variables form a tree.
- Assign *{SA=x}* ← *cycle cutset*
  - Remove any values from the other variables that are inconsistent.
  - The selected value for SA could be the wrong: have to try all of them

Nearly tree-structured CSPs



- This approach is effective if cycle cutset is small.
- Finding the smallest cycle cutset is NP-hard
  - Approximation algorithms exist
- This approach is called *cutset conditioning*.

Summary

- CSPs are a special kind of problem: states defined by values of a fixed set of variables, goal test defined by constraints on variable values
- Backtracking=depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that lead to failure.
- Constraint propagation does additional work to constrain values and detect inconsistencies.
- Structure of CSP affects its complexity. Tree structured CSPs can be solved in linear time.