Constraint Satisfaction Problems (CSPs)

Outline

- CSP?
- Backtracking search for CSPs
- Inference in CSPs
- Local search for CSPs
- Problem structure

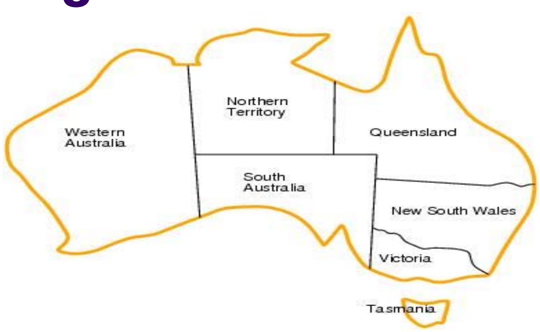
2

Constraint satisfaction problems

- What is a CSP?
  - Finite set of variables $V_1, V_2, \ldots, V_n$
  - Nonempty domain of possible values for each variable $D_{V1}, D_{V2}, \ldots D_{Vn}$
  - Finite set of constraints $C_1, C_2, \ldots, C_m$
  - Each constraint C_i specifies allowable combinations of values for subsets of variables, e.g., $V_1 \neq V_2$
  - A solution is an assignment of values to all variables that satisfies all constraints.
    - Some CSPs require a solution that maximizes an *objective function* → constrained optimization problems

3

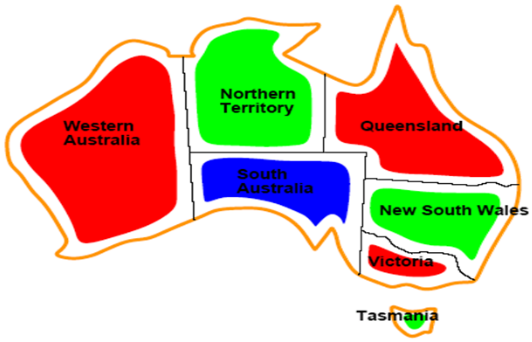
CSP example: map coloring



- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: $D_i=\{red,green,blue\}$
- Constraints: adjacent regions must have different colors.
  - E.g. $WA \neq NT$, or $(WA,NT) \in \{(red,green),(red,blue),(green,red),\ldots\}$

4

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}*

Varieties of CSPs

- Discrete variables
  - Finite domains
    - n variables with domain size $d \Rightarrow O(d^n)$ complete assignments.
    - E.g. Boolean CSPs, incl. Boolean satisfiability (NP-complete).
  - Infinite domains (integers, strings, etc.)
    - E.g. job scheduling, variables are start/end days for each job
- Continuous variables
  - e.g. start/end times for Hubble Telescope observations.
  - Linear programming

We consider finite domains

Varieties of constraints

- Unary constraints involve a single variable.
  - e.g. $SA \neq green$
  - Can be eliminated
- Binary constraints involve pairs of variables.
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables.
  - e.g. cryptarithmetic column constraints.

7

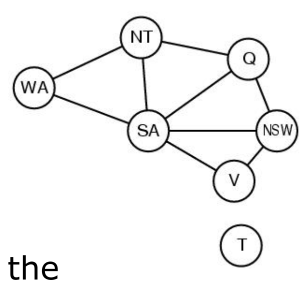
Constraint graph

- **Binary CSP**: each constraint relates two variables

• **Constraint graph** = nodes are variables, edges show constraints.

• CSP algorithms can make use of the graph structures

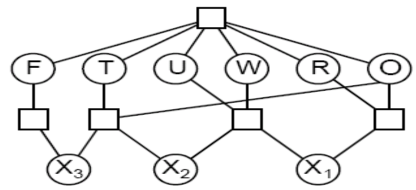
•e.g. Tasmania is an independent subproblem.



8

Example; cryptarithmetic

```
   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.

Higher-order constraints can be represented in a *constraint hypergraph*

Real-world CSPs

- Timetabling problems
  - e.g., class scheduling: which class is offered when and where by whom?
- Transportation scheduling
- Factory scheduling
- Floor planning, VLSI design
- Graph coloring, Map coloring

CSP benefits

- Standard representation with generic goal and actions
  - A state is an assignment of values to some or all variables.
  - Incremental vs. complete-state
  - Systematical vs. local search
- General purpose heuristics (no problem-specific expertise required).
- Can take advantage of the structure of constraint graph

11

CSP as a standard search problem

- A CSP can be easily expressed as a standard search problem.
- Incremental formulation
  - *A state is an assignment of values to some or all variables.*
  - *Initial State*: the empty assignment {}.
  - *Actions*: Assign value to an unassigned variable provided that there is not conflict.
  - *Goal test*: the current assignment is complete.
  - *Path cost*: irrelevant
- This is the same for all CSP's !

12

CSP as a standard search problem

- Solution is found at depth *n* (if there are *n* variables).
  - Hence depth first search can be used.
- Branching factor *b* at the top level is *nd.* $b=(n-l)d$ at depth *l*, hence $n!d^n$ leaves!
- But only d^n complete assignments.

13

Commutativity

- Variable assignments are commutative.
  - The order of any given set of actions has no effect on the outcome.
  - Example: choose colors for Australian territories one at a time
    - [WA=red then NT=green] same as [NT=green then WA=red]
- CSP search algorithms consider a single variable assignment at a time, b=d $\Rightarrow$ there are d^n leaves.

14

Backtracking search

- Depth-first search
- Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign, called **backtracking search**

15

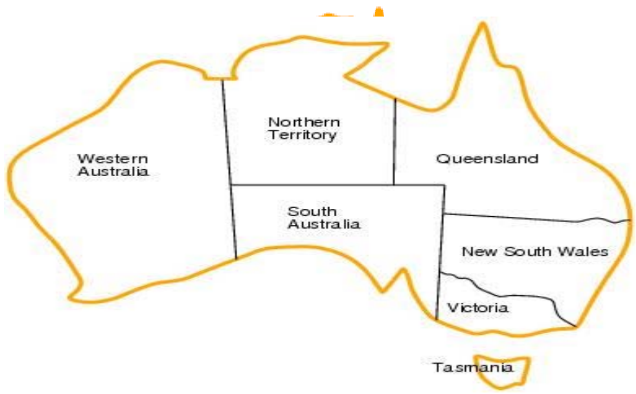
Backtracking search

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

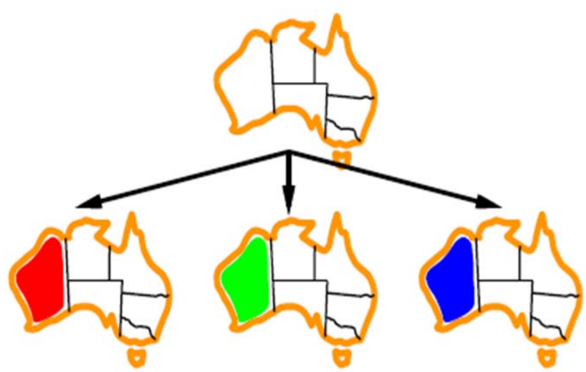
function RECURSIVE-BACKTRACKING(*assignment, csp*) **return** a solution or failure
    if *assignment* is complete **then return** *assignment*
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
    for each *value* **in** ORDER-DOMAIN-VALUES(*var, assignment, csp*) **do**
  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

16

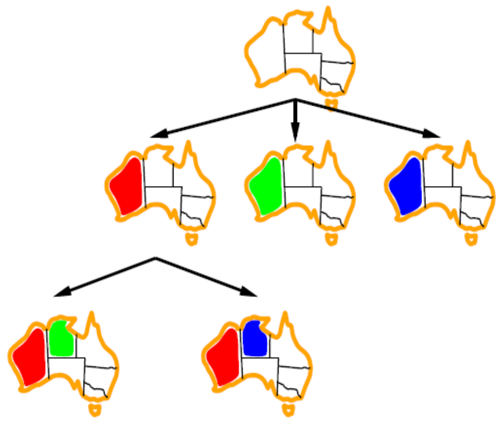
Backtracking example



Backtracking example

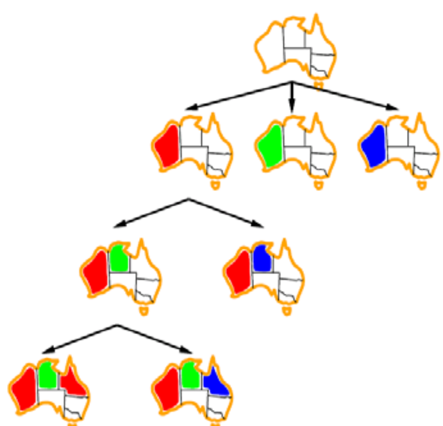


Backtracking example



19

Backtracking example



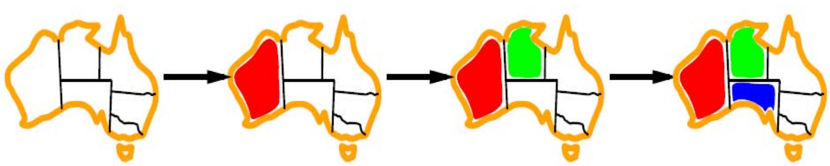
20

Improving backtracking efficiency

- Uninformed algorithm
  - general performance not good
  - Can solve n-queens for $n \approx 25$
- Previously → introduce problem-specific heuristics to improve uninformed search
- General-purpose methods without problem-specific knowledge can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Interleaving ***inference*** and Search (e.g. Sudoku)
  - Intelligent backtracking (Chapter 6.3.3)

21

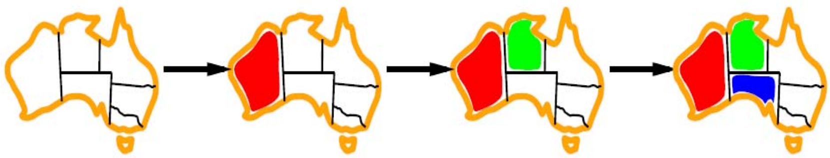
Minimum remaining values (MRV)



Which variable shall we try next?

22

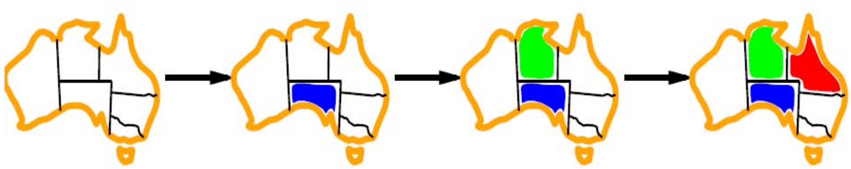
Minimum remaining values (MRV)



- A.k.a. *most constrained variable* heuristic
- *Rule*: choose variable with the fewest legal values

23

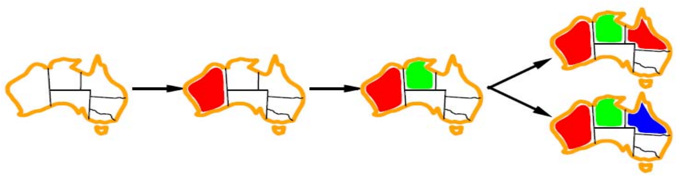
Degree heuristic



- *Rule*: select variable that is involved in the largest number of constraints on other unassigned variables. (Most constraining variable)
- Degree heuristic is very useful as a tie breaker among MRV variables.
- *In what order should its values be tried?*

24

Least constraining value

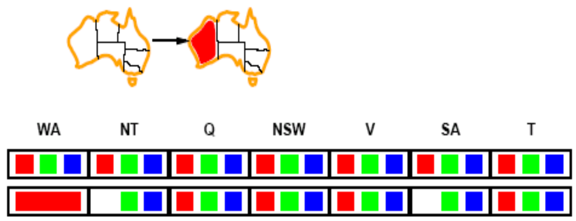


- Least constraining value heuristic
- Rule: given a variable choose the least constraining value,
  - i.e. the one that rules out the fewest values in the remaining variables

Forward checking

- Can we detect inevitable failure early?
- *Forward checking idea:* keep track of remaining legal values for unassigned variables.
  - Whenever a variable is assigned, delete from neighbors' domain any inconsistent value
  - Terminate search when any variable has no legal values.
- Provides an efficient way to incrementally compute the information that the MRV heuristic needs

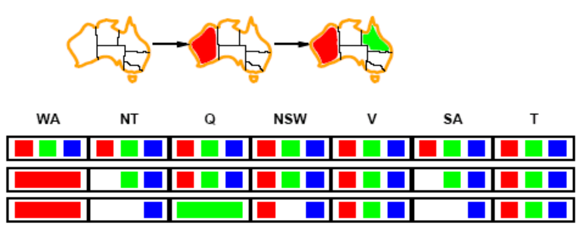
Forward checking



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

27

Forward checking

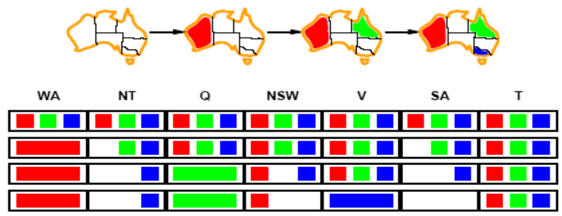


- Assign *{Q=green}*
- Effects on other variables connected by constraints with WA
  - *NT can no longer be green*
  - *NSW can no longer be green*
  - *SA can no longer be green*

(MRV heuristic will automatically select NT or SA next)

28

Forward checking

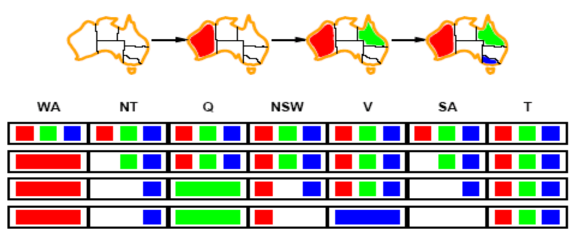


- If *V* is assigned *blue*
- Effects on other variables connected by constraints with WA
  - *SA is empty*
  - *NSW can no longer be blue*
- FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.

- Combining these heuristics makes 1000 queens feasible

29

Constraint propagation



- FC propagates information from assigned to unassigned variables but does not provide early detection for all failures.
  - NT and SA cannot both be blue!

- ***Inference*** in CSPs*:* ***Constraint propagation*** repeatedly enforces constraints locally to reduce the number of legal values for variables

30

Inference in CSPs

- CSPs could be solved by inference. E.g., Sudoku
  - repeatedly enforces constraints locally to reduce the number of legal values for variables

	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)

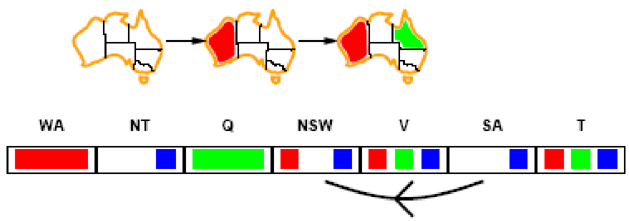
	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Inference in CSPs

- CSPs could be solved by inference.
  - repeatedly enforces constraints locally to reduce the number of legal values for variables
- Enforcing local consistency
  - Node consistency: all the values in the variable's domain satisfy unary constraints
  - Arc consistency
  - Path consistency
  - K-consistency
  - Problem-specific inference?
- Intertwined with search, or done as a preprocessing step

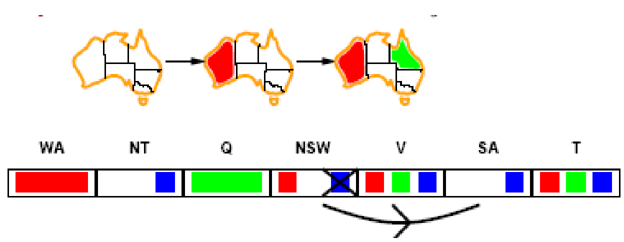
Arc consistency



- Makes each *arc consistent* (in constraint graph)
- $X \rightarrow Y$ is consistent iff
  for ***every*** value x of X there is **some** allowed y
- $SA \rightarrow NSW$ is consistent?

33

Arc consistency

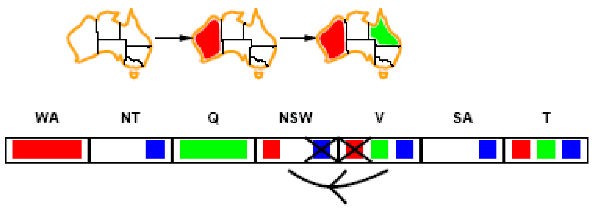


- $X \rightarrow Y$ is consistent iff
  for *every* value x of X there is some allowed y
- $NSW \rightarrow SA$ is consistent?

Arc can be made consistent by removing *blue* from *NSW*

34

Arc consistency



- Arc can be made consistent by removing *blue* from *NSW*
- If *X* loses a value, neighbors of *X* need to be rechecked
- RECHECK neighbours of *NSW*!!
  - Remove red from *V*

35

Arc consistency algorithm

function AC-3(*csp*) **return** false if an inconsistency is found and true otherwise
    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**)** ← REMOVE-FIRST(*queue*)
        if REMOVE-INCONSISTENT-VALUES(csp, X_i, X_j) **then**
            if size of D_i=0 **then return** *false*
            for each X_k **in** NEIGHBORS[X_i] – $\{X_j\}$ **do**
            add **(**X_k, X_i**)** to *queue*
    return true
function REMOVE-INCONSISTENT-VALUES(csp,X_i, X_j) **return** *true* iff we remove a value
    removed ← *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* ← *true*
    return *removed*

- Time complexity: O(cd³)

36

K-consistency

- ***Path consistency*** (3-consistency): a two-variable set {X,Y} is path consistent with respect to a third variable Z if, for every consistent assignment to {X,Y}, there is an assignment to Z that satisfies the constraints.
- ***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 be assigned to any *k*th variable
- Can be run as a preprocessing step or after each assignment during search
  - Much more expensive than FC, the extra cost is worthwhile?

37

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(*assignment*,*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 ← RRECURSIVE-BACTRACKING(*assignment, csp*)
                if *result* ≠ *failure* **then return** *result*
        remove *{var=value}* and *inferences* from *assignment*
    return *failure*

38

Local search for CSP

- Typically use complete-state representation
- For CSPs
  - A state is an assignment of values to all variables - allow unsatisfied constraints
  - operators change the value of one variable at a time
- Min-conflicts algorithm
  - Variable selection: randomly select any conflicted variable
  - Value selection: ***min-conflicts heuristic***
    - Select new value that results in a minimum number of conflicts with the other variables
    - hill-climb with *h(n)* = total number of violated constraints

39

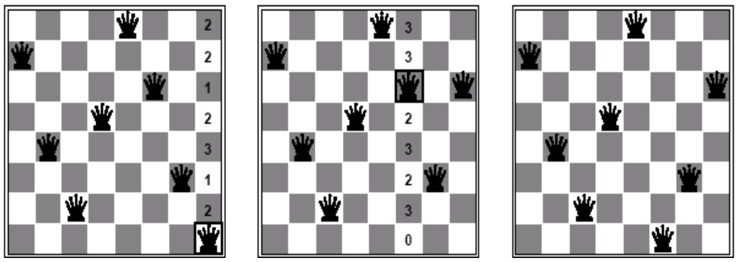
Local search for CSP

function MIN-CONFLICTS(*csp*, *max_steps*) **return** solution or failure
  inputs: *csp*, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for *csp*
  for *i* = 1 to *max_steps* **do**
    if *current* is a solution for *csp* **then return** *current*
    var ← a randomly chosen conflicted variable
    value ← the value *v* for *var* that minimize CONFLICTS(*var*,*v*,*current*,*csp*)
    set *var* = *value* in *current*
  return *failure*

40

Min-conflicts example



- A two-step solution for an 8-queens problem using min-conflicts heuristic.
- At each stage a queen is chosen for reassignment in its column.
- The algorithm moves the queen to the min-conflict square breaking ties randomly.

41

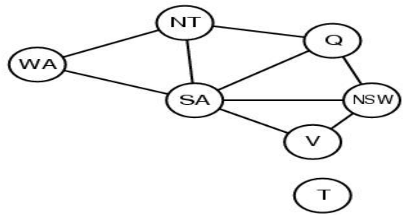
Advantages of local search

- Min-conflicts is surprisingly effective for many CSPs
  - Used to schedule observations for the Hubble Space Telescope

- Given random initial state, can solve *n*-queens in almost constant time for arbitrary *n* with high probability (e.g., *n* = 10,000,000)
  - Solving the millions-queen problem in an average of 50 steps.

42

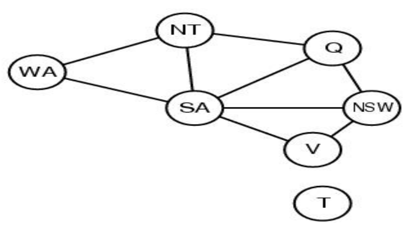
Problem structure



- *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

43

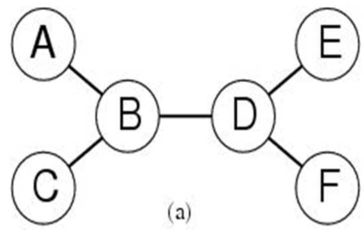
Problem structure



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

44

Tree-structured CSPs

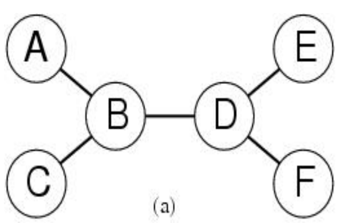


(a)

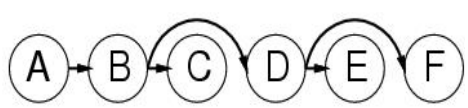
- Theorem: if the constraint graph has no loops then CSP can be solved in $O(nd^2)$ time
- Compare difference with general CSP, where worst case is $O(d^n)$

45

Tree-structured CSPs



(a)

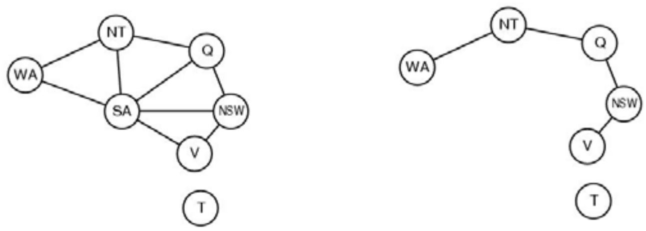


(b)

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering. (label variables from X_1 to X_n)
2. For j from n down to 2, apply arc consistency to Parent(X_j)$\rightarrow X_j$, removing values from domain of Parent(X_j) as necessary
3. For j from 1 to n assign X_j consistently with Parent(X_j)

46

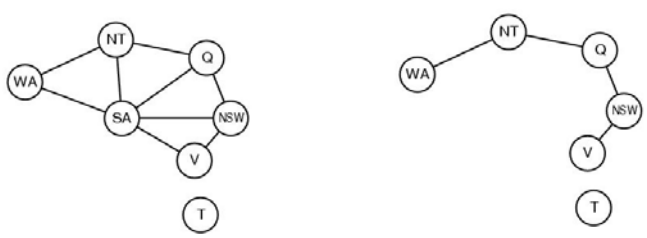
Nearly tree-structured CSPs



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

47

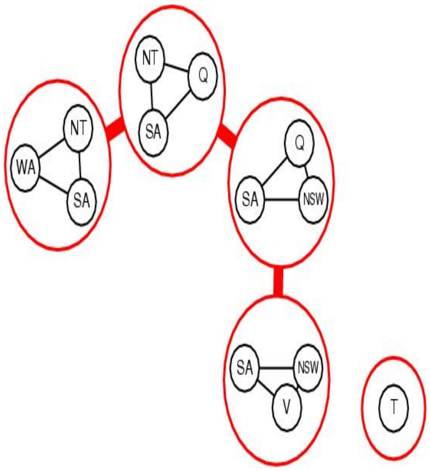
Nearly tree-structured CSPs



- This approach is worthwhile if cycle cutset is small.
  - $O(d^c (n-c)d^2)$
- Finding the smallest cycle cutset is NP-hard
  - Approximation algorithms exist
- This approach is called *cutset conditioning*.

48

Problem structure



- ***Join tree*** algorithm
- Construct a tree decomposition of the constraint graph into a set of connected subproblems.
- Each subproblem is solved independently
- Resulting solutions are combined.
- Width w: the size of the largest subproblem
  - $O(nd^w)$
- Finding minimum width (***tree width***) is NP-hard

49

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 (e.g., arc consistency) does additional work to constrain values and detect inconsistencies.
- Min-conflicts local search is usually effective in practice.
- The CSP representation allows analysis of problem structure.
  - Tree structured CSPs can be solved in linear time.

50