Set 5: Constraint Satisfaction Problems

ICS 271 Fall 2012

Rina Dechter

Outline

- **The constraint network model**
  - Variables, domains, constraints, constraint graph, solutions
- **Examples:**
  - graph-coloring, 8-queen, cryptarithmetic, crossword puzzles, vision problems,scheduling, design
- **The search space and naive backtracking,**
- **The constraint graph**
- **Consistency enforcing algorithms**
  - arc-consistency, AC-1,AC-3
- **Backtracking strategies**
  - Forward-checking, dynamic variable orderings
- **Special case: solving tree problems**
- **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

goal test is a set of *constraints* specifying
allowable combinations of values for subsets of variables

Simple example of a *formal representation language*

Allows useful *general-purpose* algorithms with more power
than standard search algorithms

Constraint Satisfaction

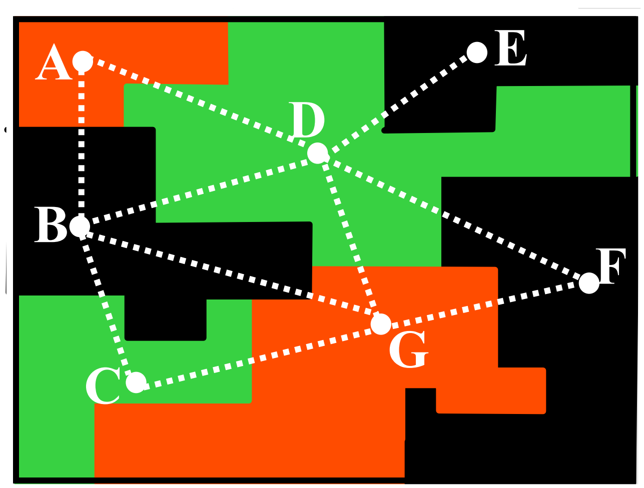
Example: map coloring

Variables - countries (A,B,C,etc.)

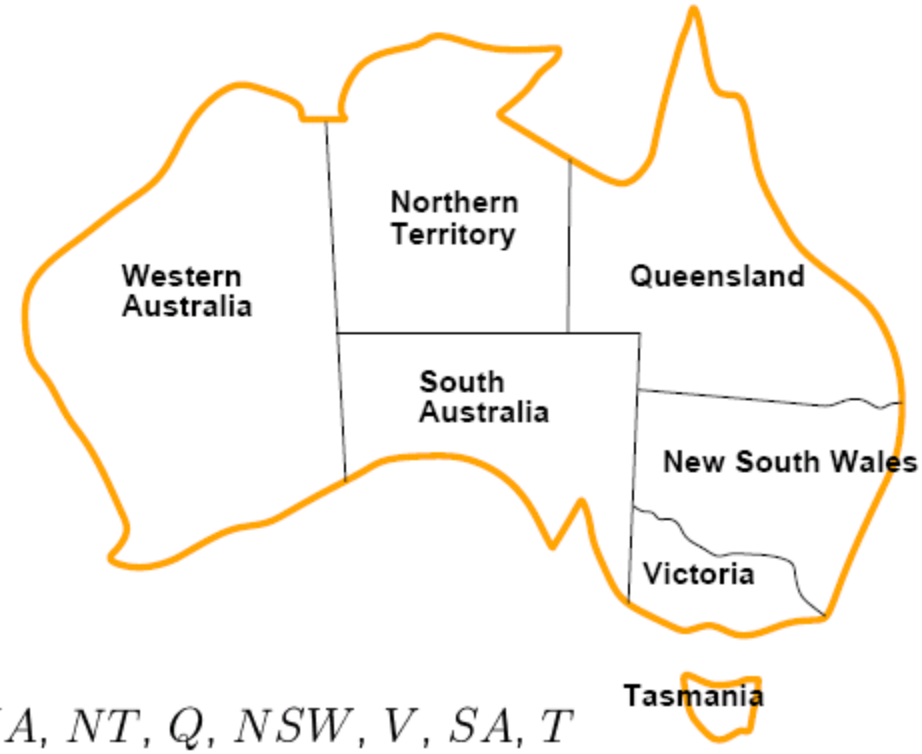
Values - colors (e.g., red, green, yellow)

Constraints: $A \neq B$, $A \neq D$, $D \neq E$, *etc.*

A	B
red	green
red	yellow
green	red
green	yellow
yellow	green
yellow	red



Example: Map-Coloring



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

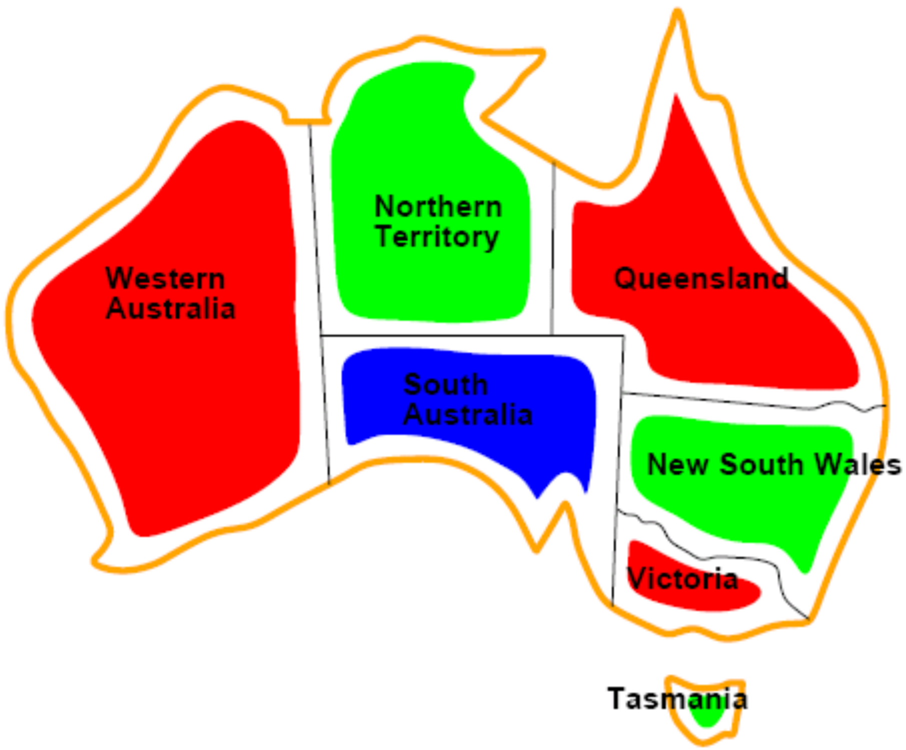
Domains $D_i = \{red, green, blue\}$

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.

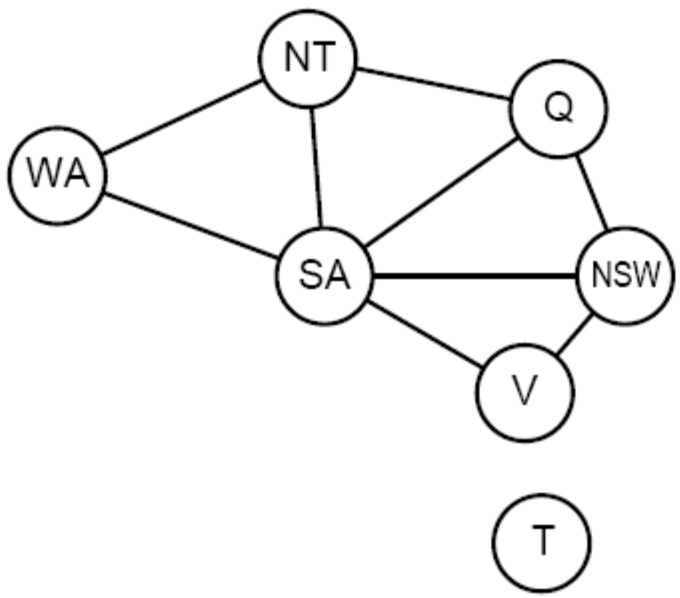


Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Constraint graph

Binary CSP: each constraint relates at most two variables

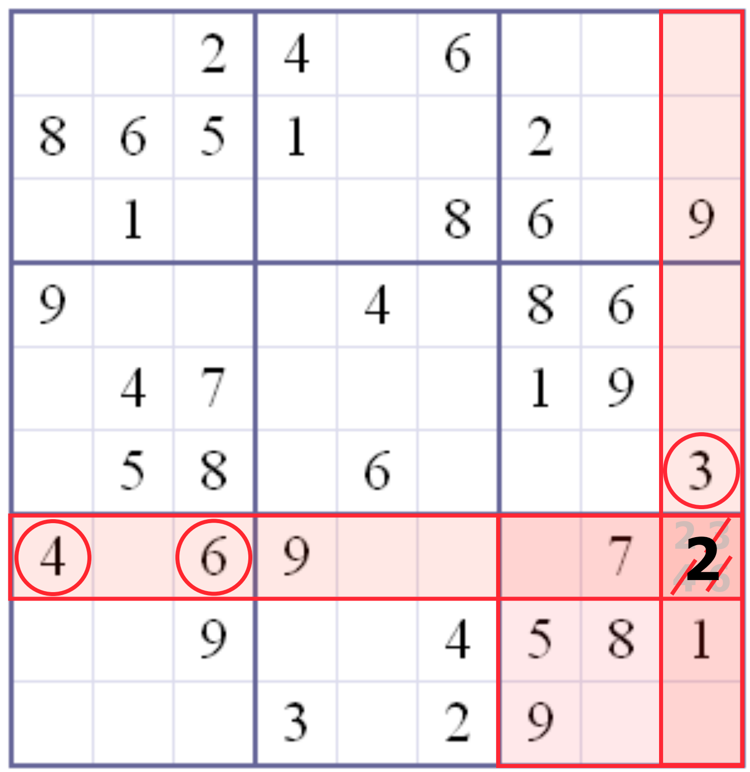
Constraint graph: nodes are variables, arcs show constraints



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

Sudoku

Constraint propagation



- **Variables: 81 slots**
- **Domains = {1,2,3,4,5,6,7,8,9}**
- **Constraints:**
  - **27 not-equal**

Each row, column and major block must be alldifferent

"Well posed" if it has unique solution: 27 constraints

Varieties of CSPs

Discrete variables

finite domains; 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
- ♢ need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- ♢ linear constraints solvable, nonlinear undecidable

Continuous variables

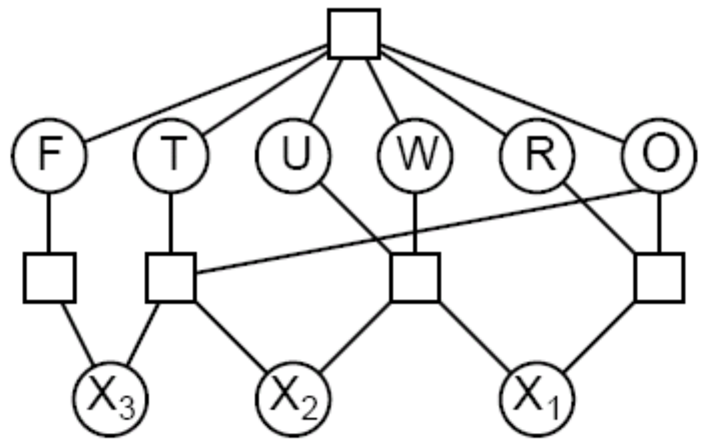
- ♢ e.g., start/end times for Hubble Telescope observations
- ♢ linear constraints solvable in poly time by LP methods

Varieties of constraints

- **Unary constraints involve a single variable,**
  - e.g., SA ≠ green
- **Binary constraints involve pairs of variables,**
  - e.g., SA ≠ WA
- **Higher-order constraints involve 3 or more variables,**
  - e.g., cryptarithmetic column constraints

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.

Real-world CSPs

Assignment problems
 e.g., who teaches what class

Timetabling problems
 e.g., which class is offered when and where?

Hardware configuration

Spreadsheets

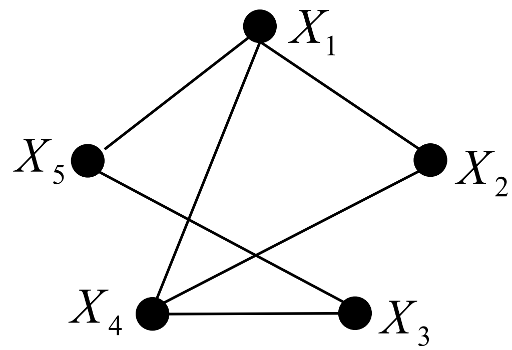
Transportation scheduling

Factory scheduling

Floorplanning

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

A network of binary constraints

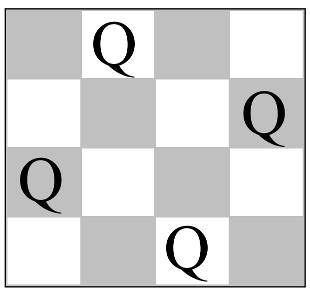


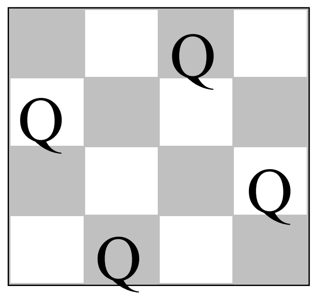
- **Variables**
  - $X_1, \ldots, X_n$
- **Domains**
  - of discrete values: $D_1, \ldots, D_n$
- **Binary constraints:**
  - R_{ij} which represent the list of allowed pairs of values, Rij is a subset of the Cartesian product: $D_i, \ldots, D_j$.
- **Constraint graph:**
  - A node for each variable and an arc for each constraint
- **Solution:**
  - An assignment of a value from its domain to each variable such that no constraint is violated.
- **A network of constraints represents the relation of all solutions.**

$$sol = \{(X_1, \ldots, X_n) \mid (x_i, x_j) \in R_{ij}, x_i \in D_i, X_j \in D_j\}$$

Example 1: The 4-queen problem

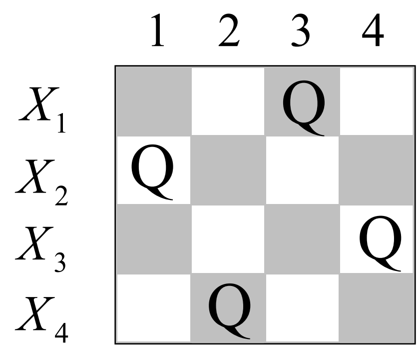
Place 4 Queens on a chess board of 4x4 such that no two queens reside in the same row, column or diagonal.





Standard CSP formulation of the problem:

- **Variables**: each row is a variable.



- **Domains**: $D_i = \{1,2,3,4\}$.

- **Constraints**: There are $\binom{4}{2} = 6$ constraints involved:

$R_{12} = \{(1,3)(1,4)(2,4)(3,1)(4,1)(4,2)\}$

$R_{13} = \{(1,2)(1,4)(2,1)(2,3)(3,2)(3,4)(4,1)(4,3)\}$

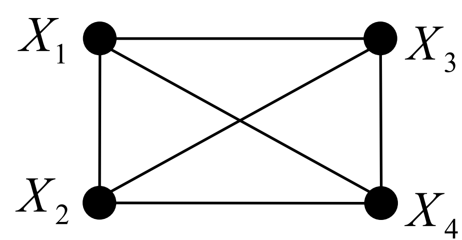
$R_{14} = \{(1,2)(1,3)(2,1)(2,3)(2,4)(3,1)(3,2)(3,4)(4,2)(4,3)\}$

$R_{23} = \{(1,3)(1,4)(2,4)(3,1)(4,1)(4,2)\}$

$R_{24} = \{(1,2)(1,4)(2,1)(2,3)(3,2)(3,4)(4,1)(4,3)\}$

$R_{34} = \{(1,3)(1,4)(2,4)(3,1)(4,1)(4,2)\}$

- **Constraint Graph** :



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.
⇒ fail if no legal assignments (not fixable!)

◊ Goal test: the current assignment is complete

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

Backtracking search

Variable assignments are commutative, i.e.,

$[WA = red$ then $NT = green]$ same as $[NT = green$ then $WA = red]$

Only need to consider assignments to a single variable at each node

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

Depth-first search for CSPs with single-variable assignments
is called backtracking search

Backtracking search is the basic uninformed algorithm for CSPs

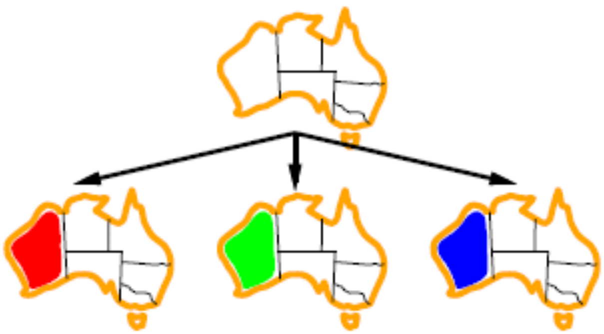
Can solve n-queens for $n \approx 25$

The search space

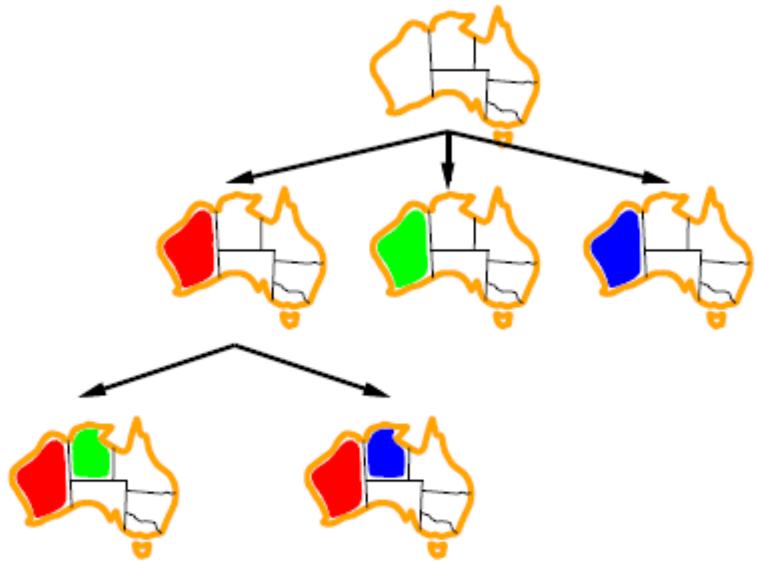
- **Definition: given an ordering of the variables** $X_1, \ldots, X_n$
  - **a state:**
    - is an assignment to a subset of variables that is consistent.
  - **Operators:**
    - add an assignment to the next variable that does not violate any constraint.
  - **Goal state**:
    - a consistent assignment to **all** the variables.

Backtracking example

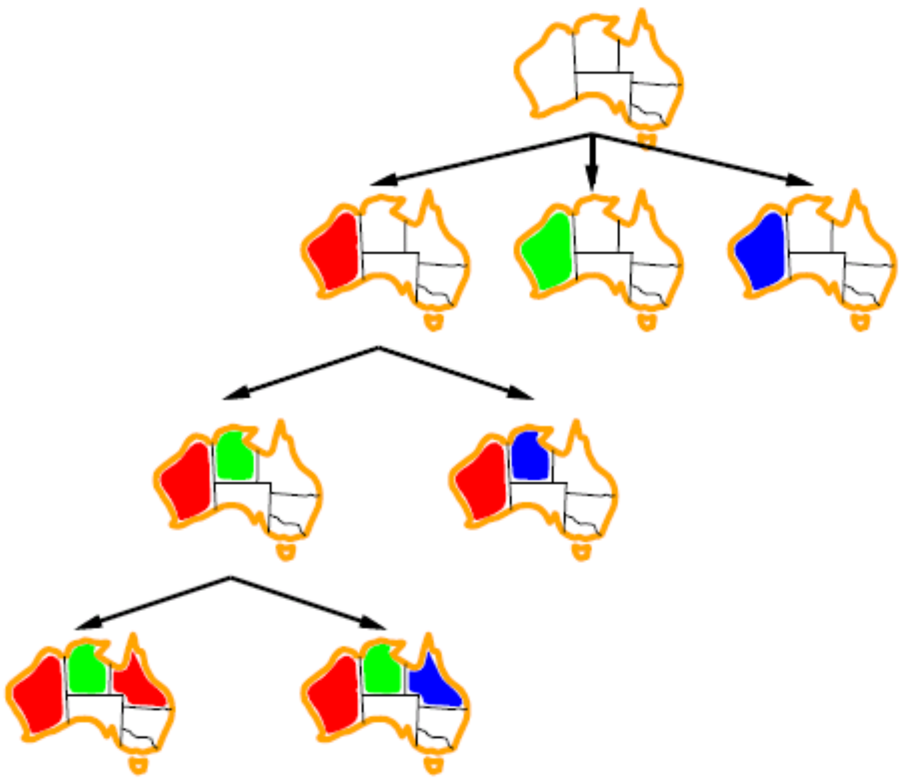
Backtracking example



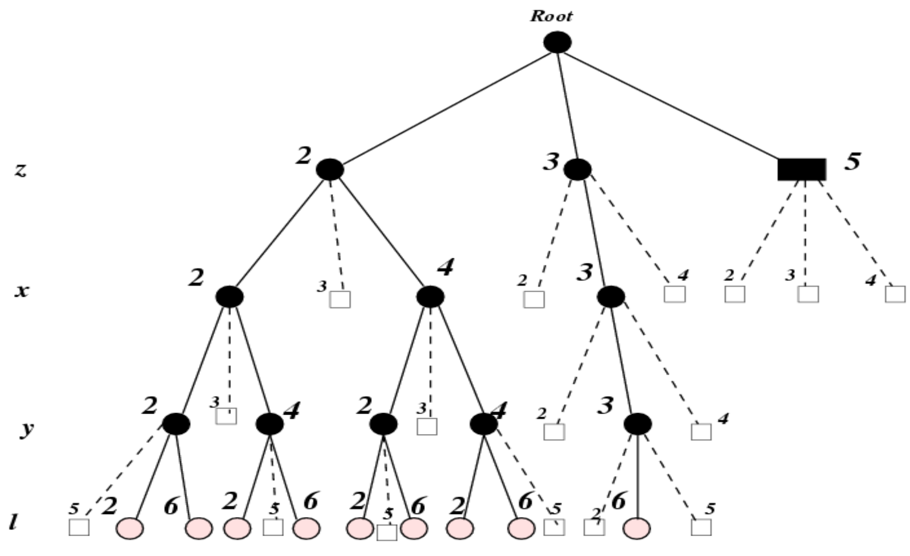
Backtracking example



Backtracking example

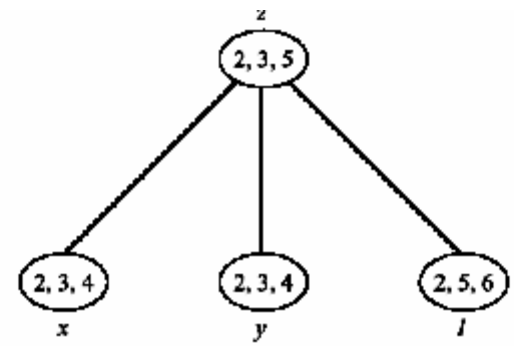


The effect of variable ordering

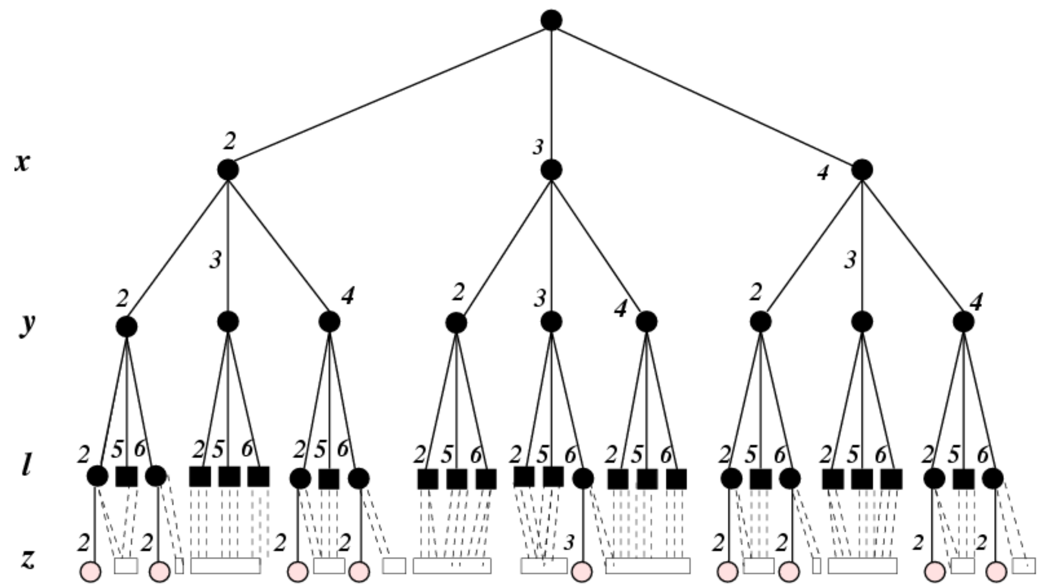


(b)

z divides x, y and t



(a)



(c)

Backtracking

```
procedure BACKTRACKING
Input: A constraint network P = (X, D, C).
Output: Either a solution, or notification that the network is inconsistent.

   i ← 1                          (initialize variable counter)
   D'_i ← D_i                     (copy domain)
   while 1 ≤ i ≤ n
      instantiate x_i ← SELECTVALUE
      if x_i is null              (no value was returned)
         i ← i − 1                (backtrack)
      else
         i ← i + 1                (step forward)
         D'_i ← D_i
   end while
   if i = 0
      return "inconsistent"
   else
      return instantiated values of {x_1, ..., x_n}
end procedure

subprocedure SELECTVALUE   (return a value in D'_i consistent with ā_{i−1})

   while D'_i is not empty
      select an arbitrary element a ∈ D'_i, and remove a from D'_i
      if CONSISTENT(ā_{i−1}, x_i = a)
         return a
   end while
   return null                    (no consistent value)
end procedure
```

Figure 5.4: The backtracking algorithm.

- Complexity of extending a partial solution:
  - Complexity of consistent: O(e log t), t bounds #tuples, e bounds #constraints
  - Complexity of selectvalue: O(e k log t), k bounds domain size

A coloring problem

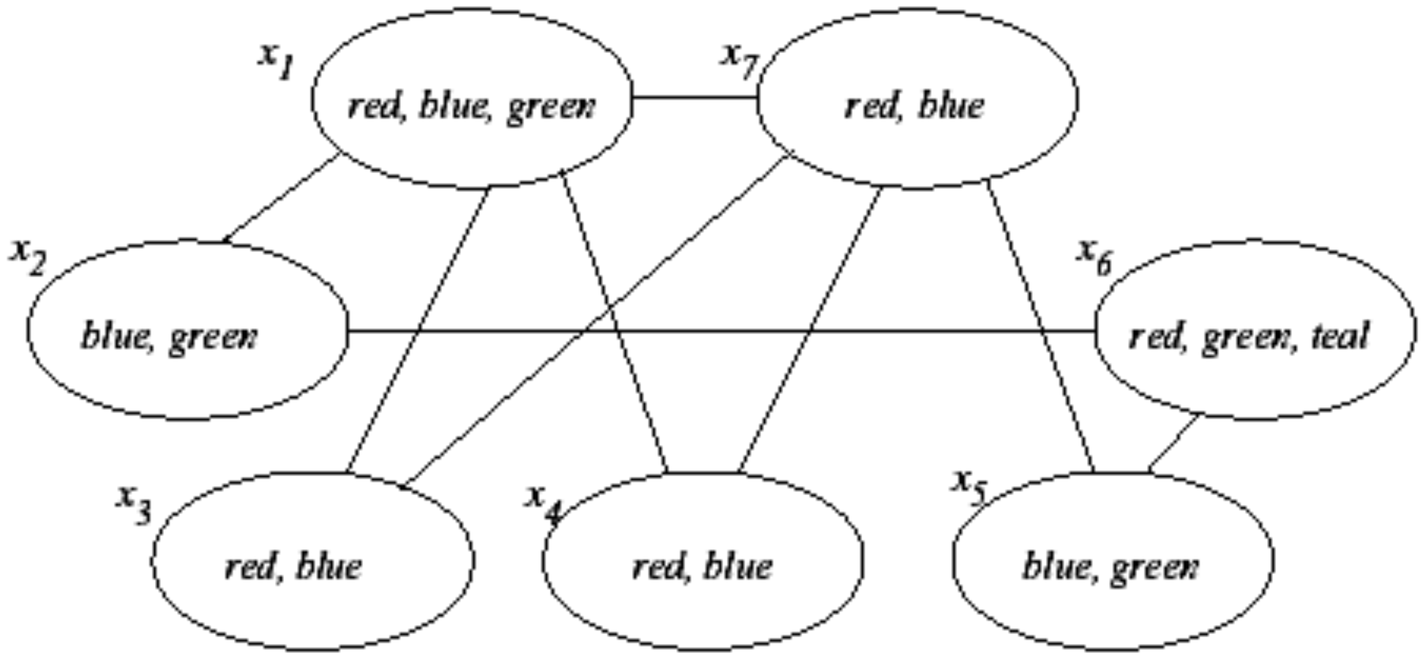
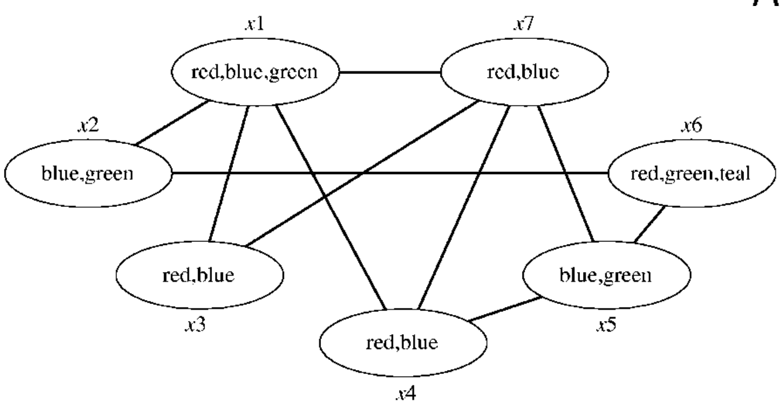
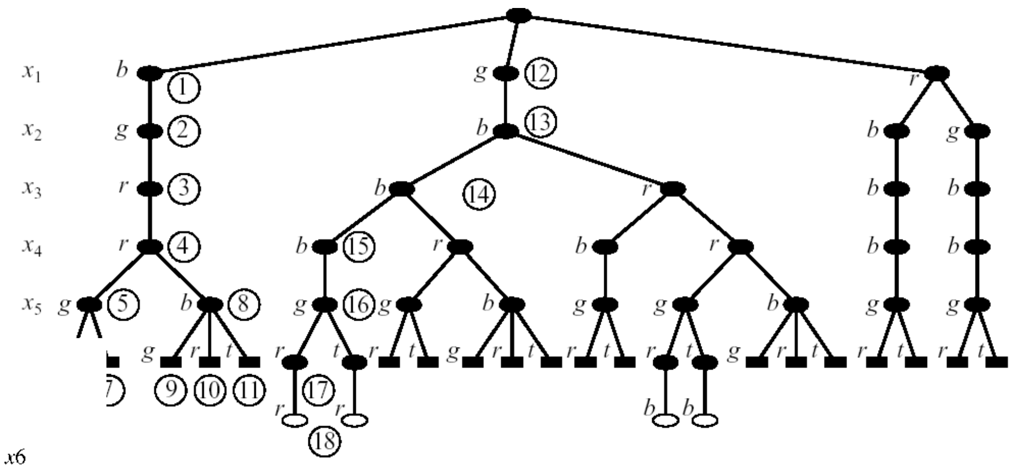


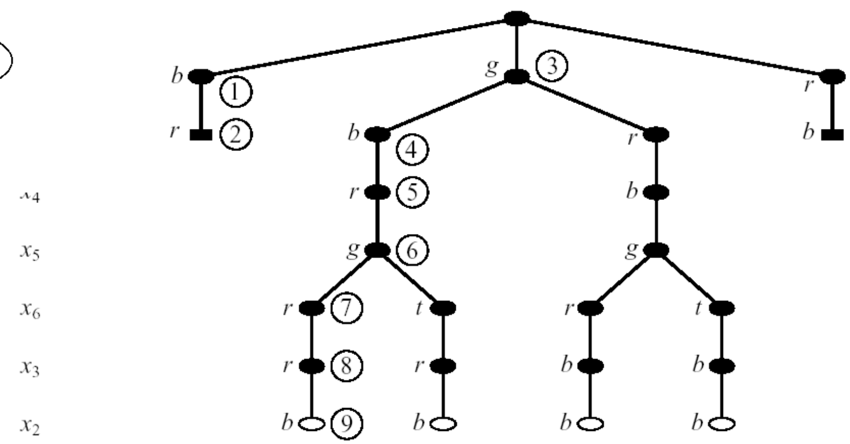
Figure 5.3: A coloring problem with variables $(x_1, x_2, \ldots, x_7)$. The domain of each variable is written inside the corresponding node. Each arc represents the constraint that the two variables it connects must be assigned different colors.

Backtracking Search for a Solution



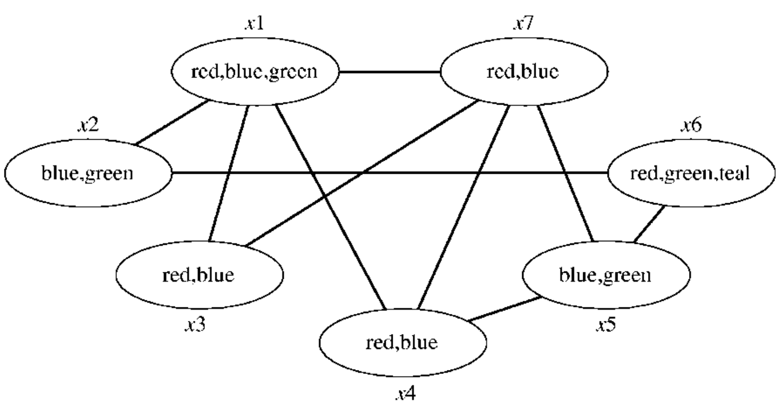


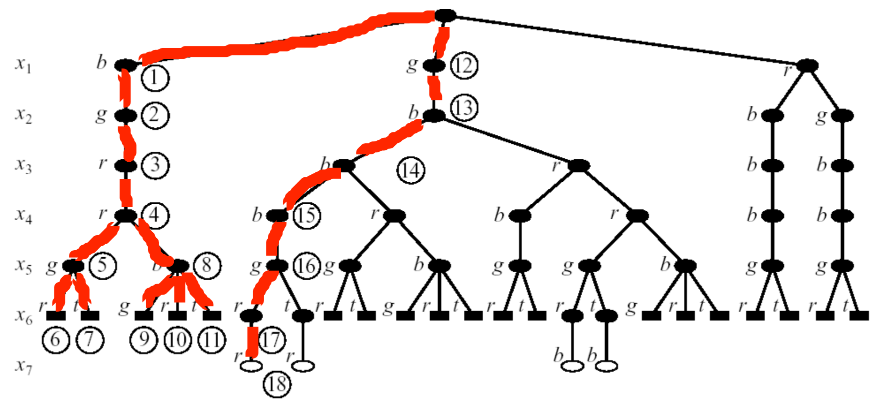
(a)



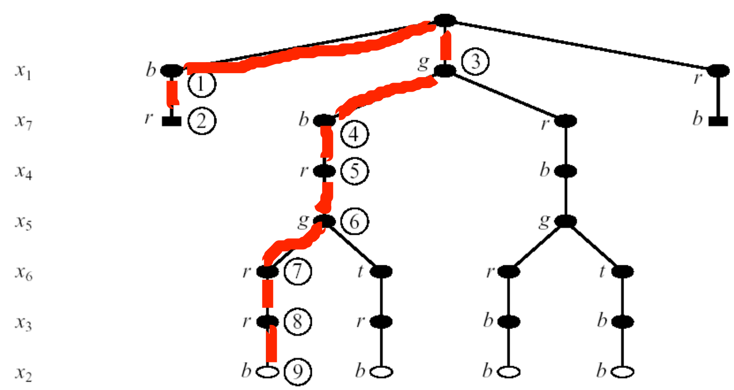
(b)

Backtracking Search for a Solution



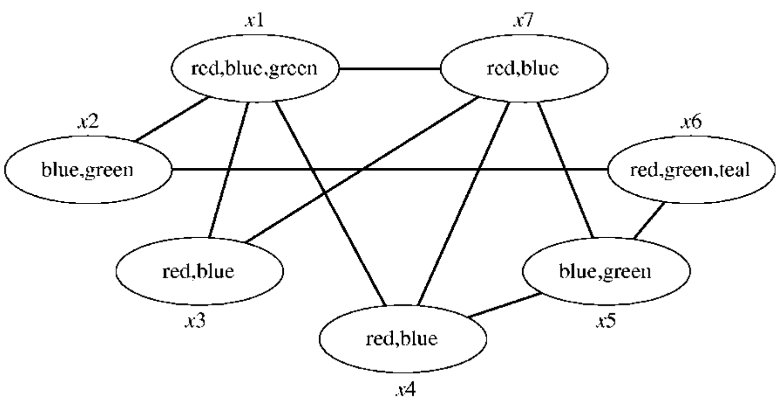


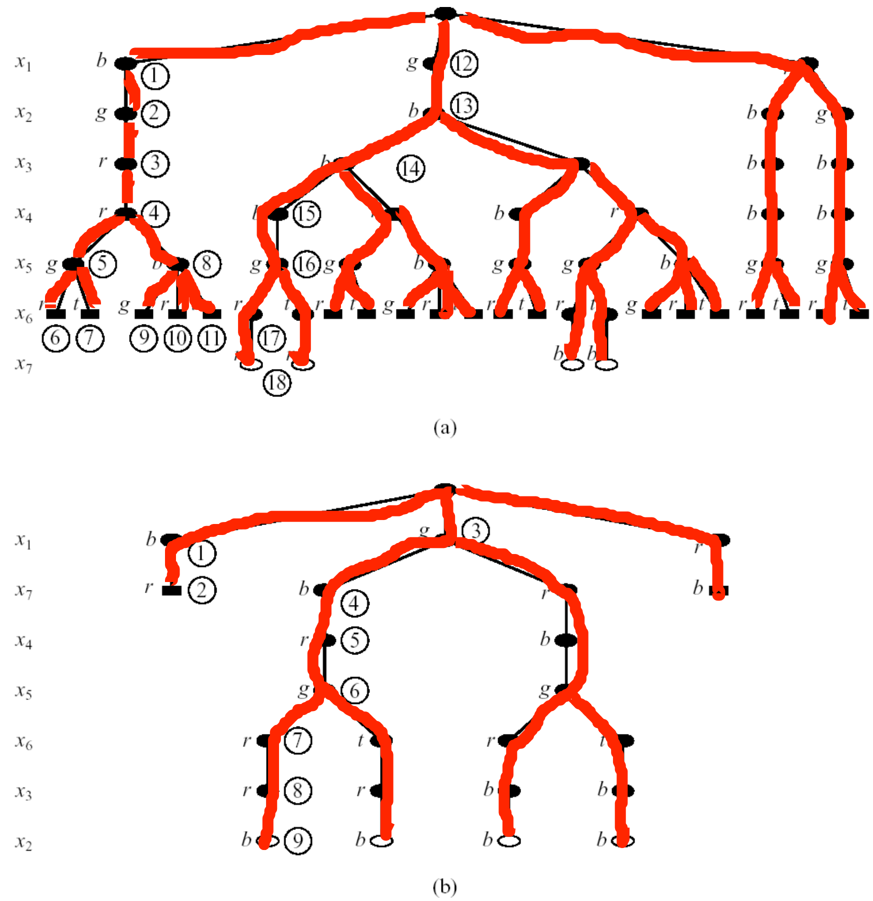
(a)



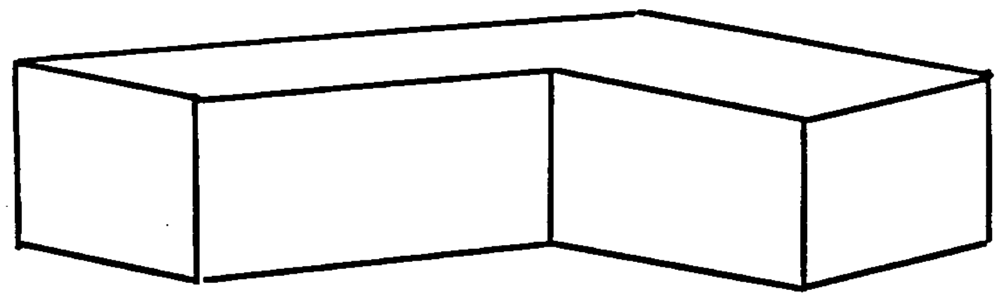
(b)

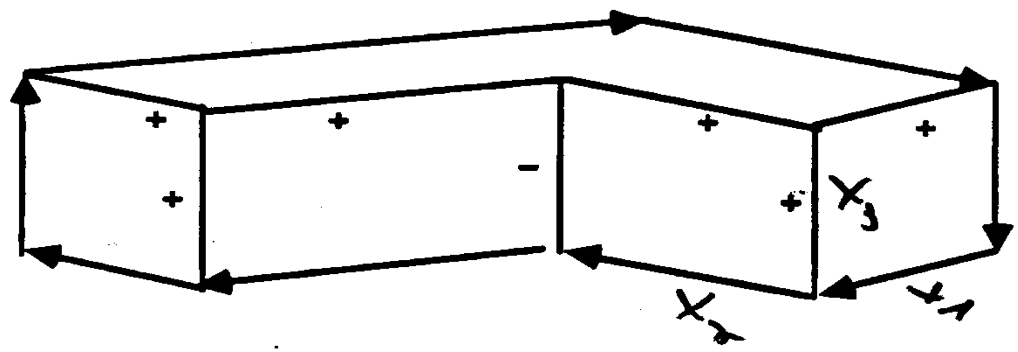
Backtracking Search for All Solutions

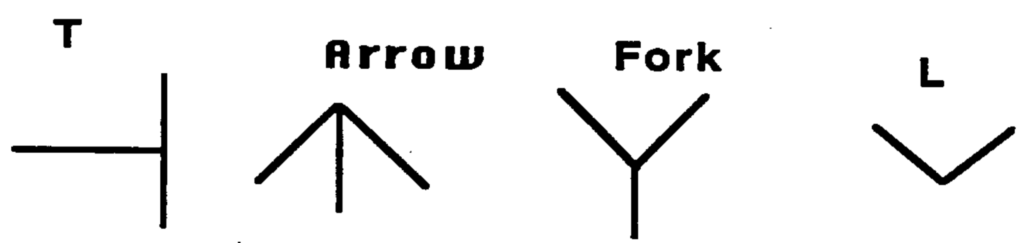




Line drawing Interpretations

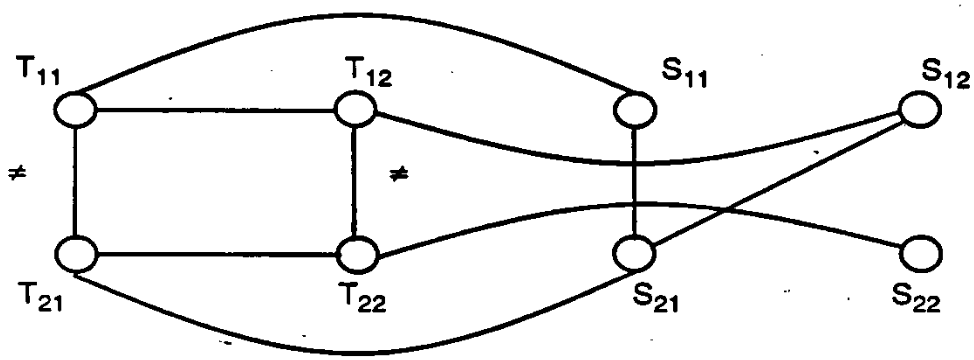






Class scheduling/Timetabling

- Teachers, Subjects, Classrooms, Time-slots.
- Constraints: - A teacher teaches a subset of subjects,
  - Subjects are taught at certain classrooms,
  - A teacher prefers teaching in the morning.
- Task: Assign a teacher and a subject to each class at each time slot, s.t. teachers' happiness is maximized.



T_{ij} - teacher at class C_i at time t_j $D(T_{ij})$ = teachers

S_{ij} - subject taught at class C_i at time t_j

Domain: subjects

The Minimal network: Example: the 4-queen problem

	Q		
			Q
Q			
		Q	

← *two solutions* →

		Q	
Q			
			Q
	Q		

	1	2	3	4
X_1 →		Q		
X_2 →				Q
X_3 →	Q			
X_4 →			Q	

$$
\begin{aligned}
R_{12} &= \{\ (1,3)\ (1,4)\ (2,4)\ (3,1)\ (4,1)\ (4,2)\ \} \\
R_{13} &= \{\ (1,2)\ (1,4)\ (2,1)\ (2,3)\ (3,2)\ (3,4)\ (4,1)\ (4,3)\ \} \\
R_{14} &= \{\ (1,2)\ (1,3)\ (2,1)\ (2,3)\ (2,4)\ (3,1)\ (3,2)\ (3,4)\ (4,2)\ (4,3)\ \} \\
R_{23} &= \{\ (1,3)\ (1,4)\ (2,4)\ (3,1)\ (4,1)\ (4,2)\ \} \\
R_{24} &= \{\ (1,2)\ (1,4)\ (2,1)\ (2,3)\ (3,2)\ (3,4)\ (4,1)\ (4,3)\ \} \\
R_{34} &= \{\ (1,3)\ (1,4)\ (2,4)\ (3,1)\ (4,1)\ (4,2)\ \}
\end{aligned}
$$

$$
\textit{the solution}\ \ \rho = \left(\begin{array}{cccc} X_1 & X_2 & X_3 & X_4 \\ \hline 2 & 4 & 1 & 3 \\ 3 & 1 & 4 & 2 \end{array} \right)
$$

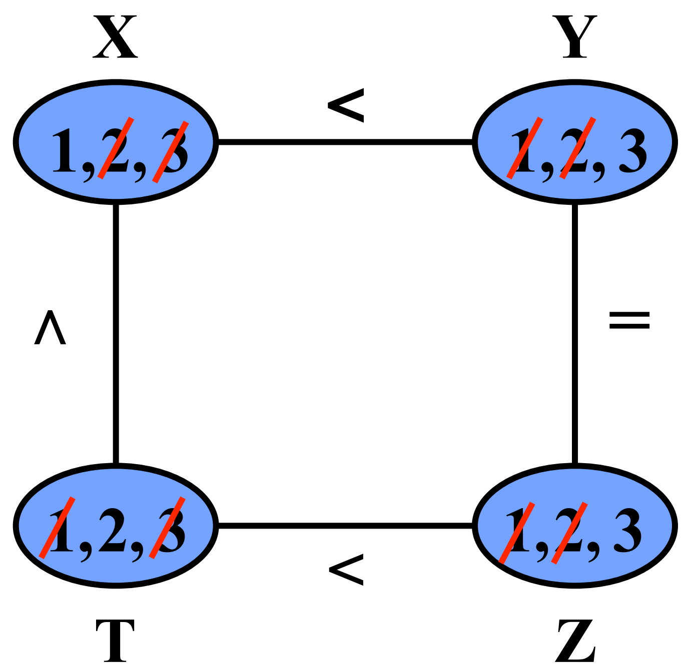
$$
M_\rho = proj(\rho) = \left\{ \begin{aligned}
M_{12} &= \{\ (2,4)\ (3,1)\ \} \\
M_{13} &= \{\ (2,1)\ (3,4)\ \} \\
M_{14} &= \{\ (2,3)\ (3,2)\ \} \\
M_{23} &= \{\ (1,4)\ (4,1)\ \} \\
M_{24} &= \{\ (1,2)\ (4,3)\ \} \\
M_{34} &= \{\ (1,3)\ (4,2)\ \}
\end{aligned} \right\}
$$

Approximation algorithms

- Arc-consistency (Waltz, 1972)
- Path-consistency (Montanari 1974, Mackworth 1977)
- I-consistency (Freuder 1982)
- Transform the network into smaller and smaller networks.

Arc-consistency

1 ≤ X, Y, Z, T ≤ 3
X < Y
Y = Z
T < Z
X ≤ T



Arc-consistency

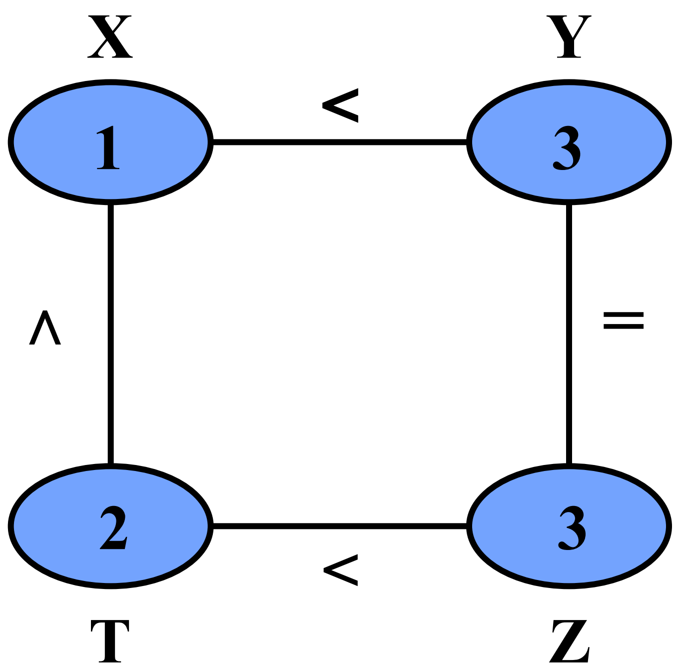
$1 \leq X, Y, Z, T \leq 3$

$X < Y$

$Y = Z$

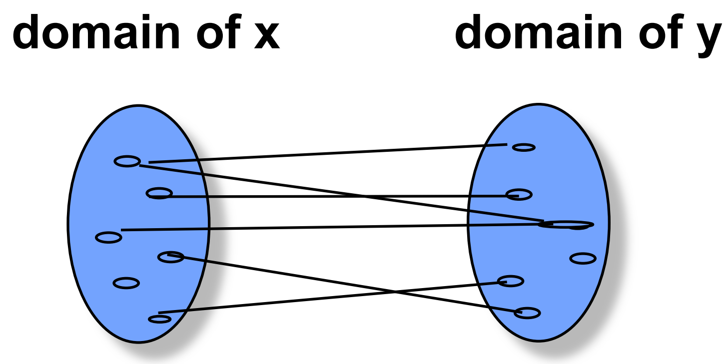
$T < Z$

$X \leq T$



- Incorporated into backtracking search
- <u>Constraint programming languages</u> powerful approach for modeling and solving combinatorial optimization problems.

Arc-consistency algorithm



Arc (X_i, X_j) is arc-consistent if for any value of X_i there exist a matching value of X_i

Algorithm Revise (X_i, X_j) makes an arc consistent

Begin

1. For each a in D_i if there is no value b in D_j that matches a then delete a from the D_j.

End.

Revise is $O(k^2)$, *k* is the number of value in each domain.

Algorithm AC-3

- **Begin**
  - 1. Q <--- put all arcs in the queue in both directions
  - 2. While Q is not empty do,
  - 3. Select and delete an arc (X_i, X_j) from the queue Q
    - 4. Revise (X_i, X_j)
    - 5. If Revise cause a change then add to the queue all arcs that touch X_i (namely (X_i, X_m) and (X_l, X_i)).
  - 6. end-while
- **End**
- **Complexity:**
  - Processing an arc requires O(k^2) steps
  - The number of times each arc can be processed is 2·k
  - Total complexity is $O(ek^3)$

Sudoku

Constraint propagation

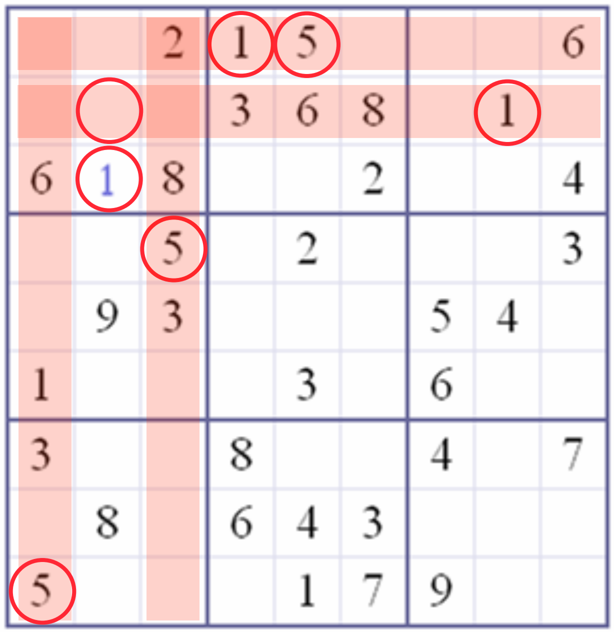
		2	4		6			
8	6	5	1			2		
	1				8	6		9
9				4		8	6	
	4	7				1	9	
	5	8		6				3
4		6	9				7	2
		9			4	5	8	1
			3		2	9		

- **Variables**: 81 slots
- **Domains** = {1,2,3,4,5,6,7,8,9}
- **Constraints**:
  - 27 not-equal

Each row, column and major block must be alldifferent

"Well posed" if it has unique solution: 27 constraints

Sudoku



Path-consistency or 3-consistency

4-consistency and i-consistency in geeral

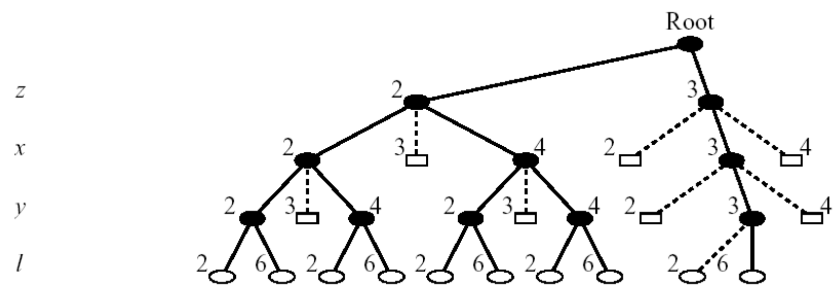
Each row, column and major block must be alldifferent

“Well posed” if it has unique solution

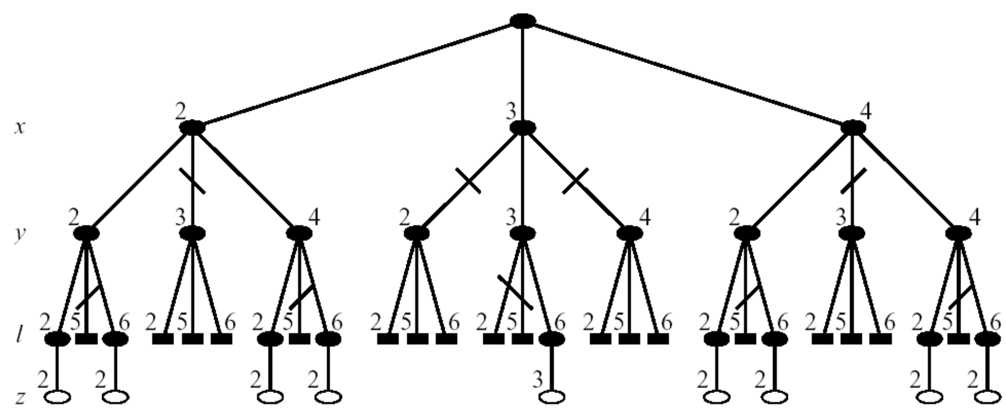
The Effect of Consistency Level

- **After arc-consistency z=5 and l=5 are removed**

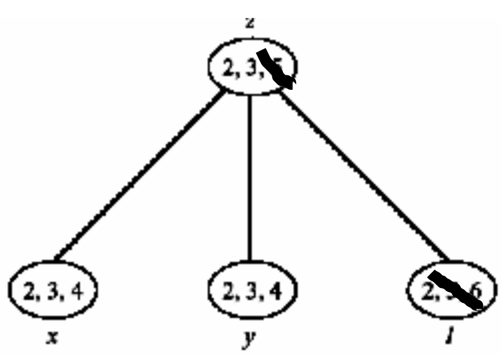
- **After path-consistency**
  - R’_zx
  - R’_zy
  - R’_zl
  - R’_xy
  - R’_xl
  - R’_yl



(a)



(b)



(a)

Tighter networks yield smaller search spaces

Improving Backtracking O(exp(n))

- **Before search: (reducing the search space)**
  - Arc-consistency, path-consistency, i-consistency
  - Variable ordering (fixed)
- **During search**:
  - **Look-ahead schemes**:
    - Value ordering/pruning (*choose a least restricting value*),
    - Variable ordering ***(Choose the most constraining variable)***
  - **Look-back schemes**:
    - Backjumping
    - Constraint recording
    - Dependency-directed backtracking

Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

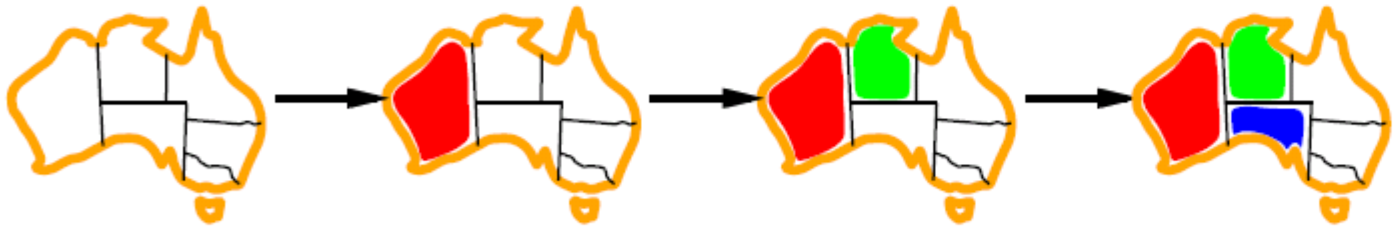
1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

Look-ahead: Variable and value orderings

- **Intuition:**
  - Choose value least likely to yield a dead-end
  - Choose a variable that will detect failures early
  - Approach: apply propagation at each node in the search tree
- **Forward-checking**
  - (check each unassigned variable separately
- **Maintaining arc-consistency (MAC)**
  - (apply full arc-consistency)

Most constrained variable

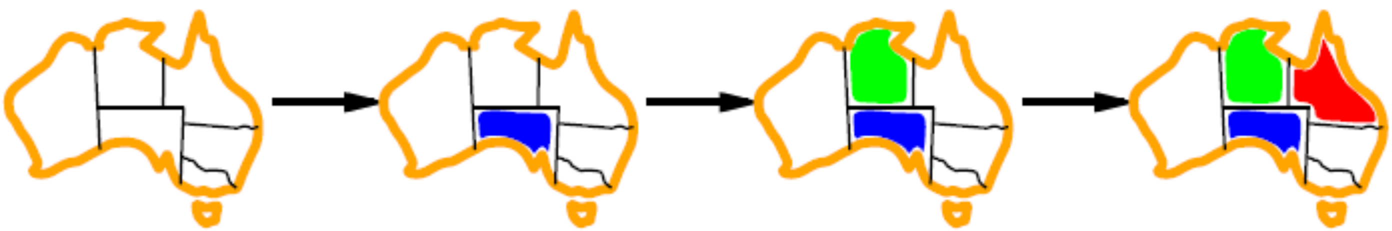
Most constrained variable:
choose the variable with the fewest legal values



Most constraining variable

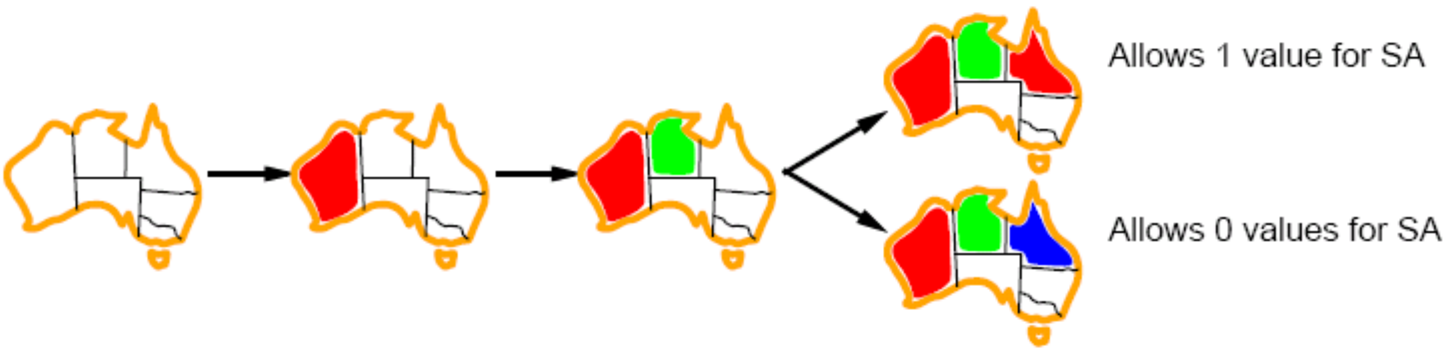
Tie-breaker among most constrained variables

Most constraining variable:
choose the variable with the most constraints on remaining variables



Least constraining value

Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables



Combining these heuristics makes 1000 queens feasible

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Forward checking

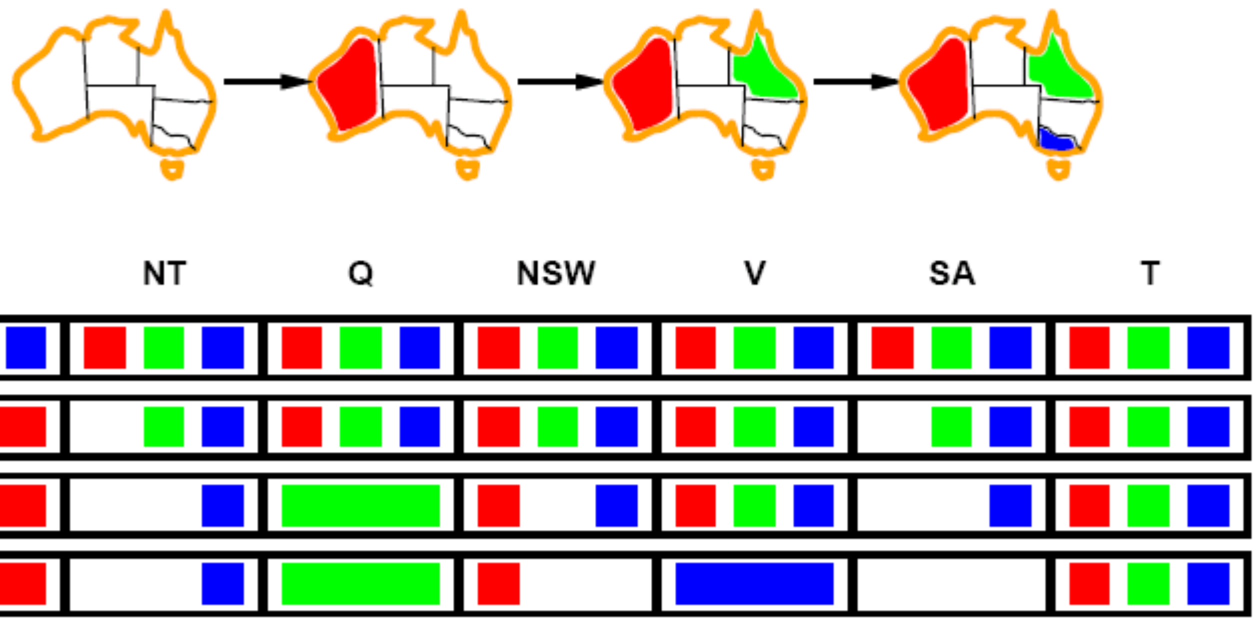
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

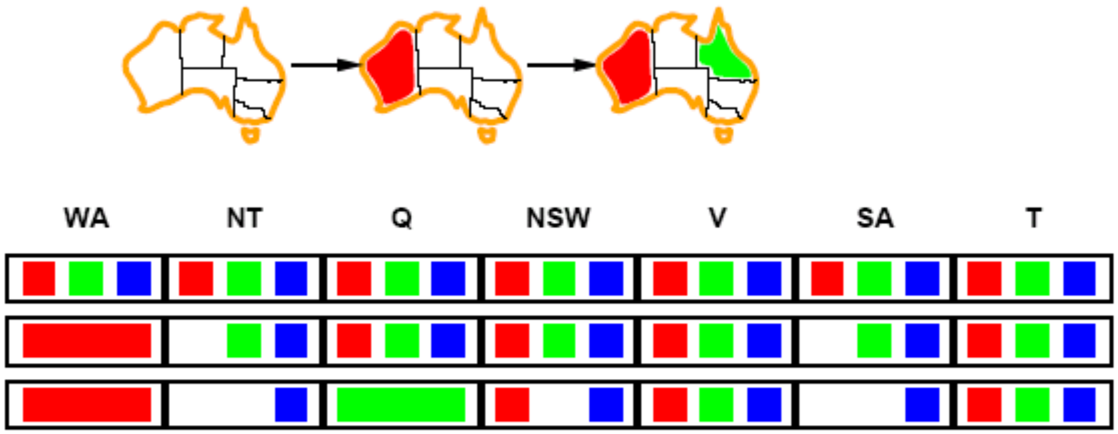
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



NT and SA cannot both be blue!

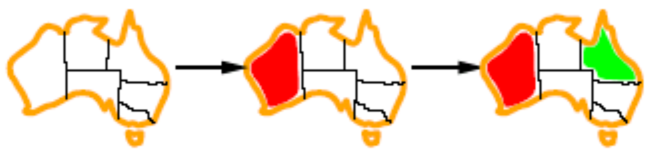
Constraint propagation repeatedly enforces constraints locally

Arc consistency

Simplest form of propagation makes each arc consistent

$X \to Y$ is consistent iff

for *every* value x of X there is *some* allowed y

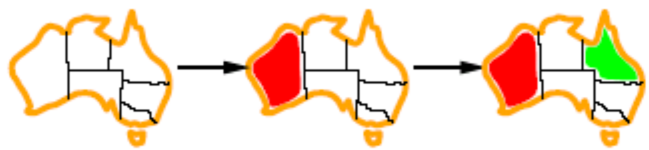


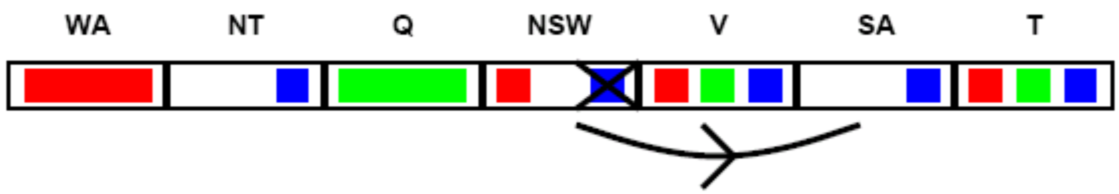
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for *every* value x of X there is *some* allowed y



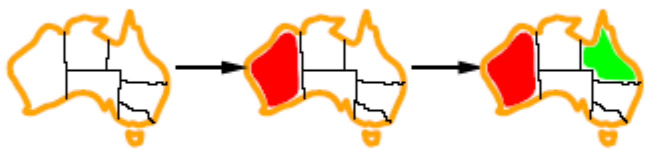


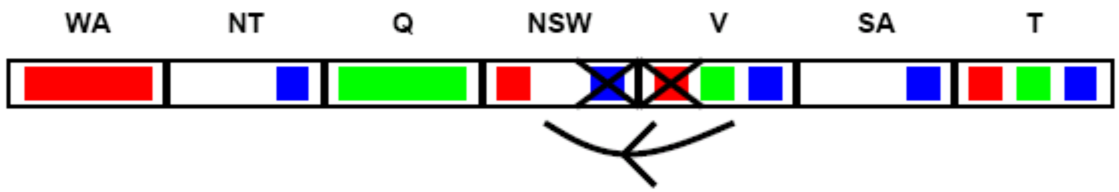
Arc consistency

Simplest form of propagation makes each arc consistent

$X \to Y$ is consistent iff

for *every* value x of X there is *some* allowed y





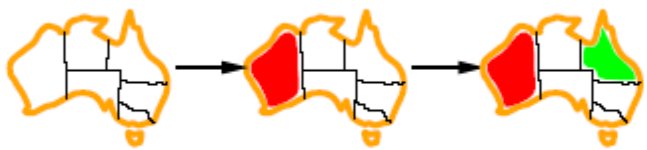
If X loses a value, neighbors of X need to be rechecked

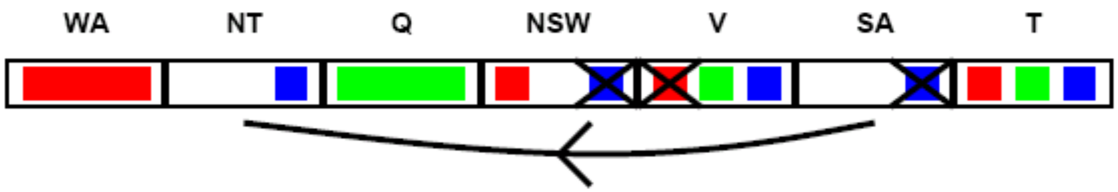
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for *every* value x of X there is *some* allowed y



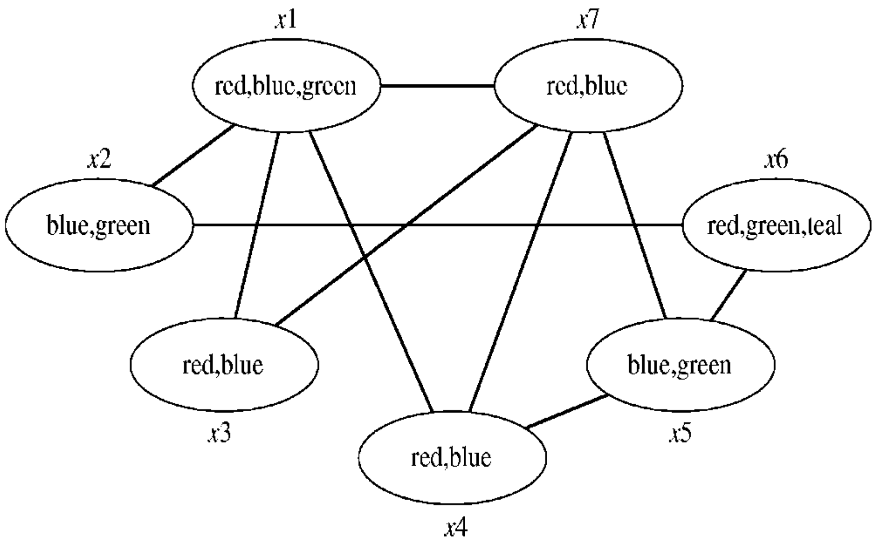


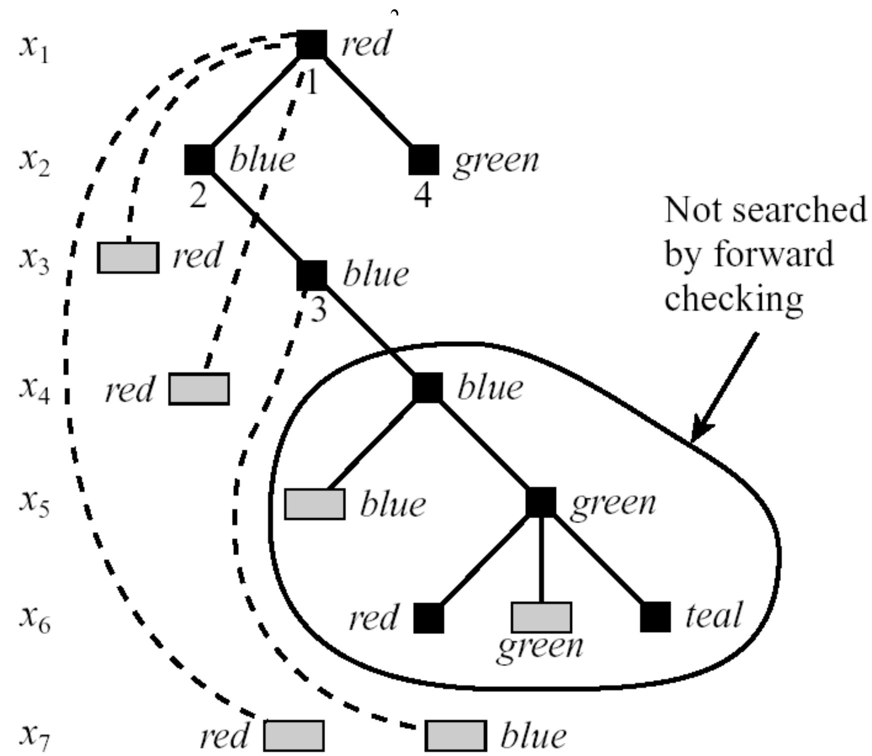
If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Forward-checking on Graph-coloring





FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

Algorithm DVO (DVFC)

```
procedure DVFC
Input: A constraint network R = (X, D, C)
Output: Either a solution, or notification that the network is inconsistent.
    D'_i ← D_i for 1 ≤ i ≤ n          (copy all domains)
    i ← 1                              (initialize variable counter)
          s = min_{i<j≤n} |D'_j|       (find future var with smallest domain)
          x_{i+1} ← x_s (rearrange variables so that x_s follows x_i)
    while 1 ≤ i ≤ n
      instantiate x_i ← SELECTVALUE-FORWARD-CHECKING
      if x_i is null                   (no value was returned)
        reset each D' set to its value before x_i was last instantiated
        i ← i − 1                      (backtrack)
      else
        if i < n
        i ← i + 1                      (step forward to x_s)
          s = min_{i<j≤n} |D'_j|       (find future var with smallest domain)
          x_{i+1} ← x_s (rearrange variables so that x_s follows x_i)
        i ← i + 1                      (step forward to x_s)
    end while
    if i = 0
      return "inconsistent"
    else
      return instantiated values of {x_1, . . . , x_n}
end procedure
```

Figure 5.12: The DVFC algorithm. It uses the SELECTVALUE-FORWARD-CHECKING sub-procedure given in Fig. 5.8.

Propositional Satisfiability

Example: party problem

- **If Alex goes, then Becky goes:** $\mathbf{A} \rightarrow \mathbf{B}$ (or, $\neg \mathbf{A} \vee \mathbf{B}$)
- **If Chris goes, then Alex goes:** $\mathbf{C} \rightarrow \mathbf{A}$ (or, $\neg \mathbf{C} \vee \mathbf{A}$)
- Query:

Is it possible that Chris goes to the party but Becky does not?

$\Updownarrow$

Is *propositional theory*

$$\varphi = \{\neg \mathbf{A} \vee \mathbf{B},\ \neg \mathbf{C} \vee \mathbf{A},\ \neg \mathbf{B},\ \mathbf{C}\} \text{ satisfiable?}$$

Unit Propagation

- **Arc-consistency for cnfs.**
- **Involve a single clause and a single literal**
- **Example:** $(A, \neg B, C) \wedge B \longrightarrow (A,C)$

Look-ahead for SAT
(Davis-Putnam, Logeman and Laveland, 1962)

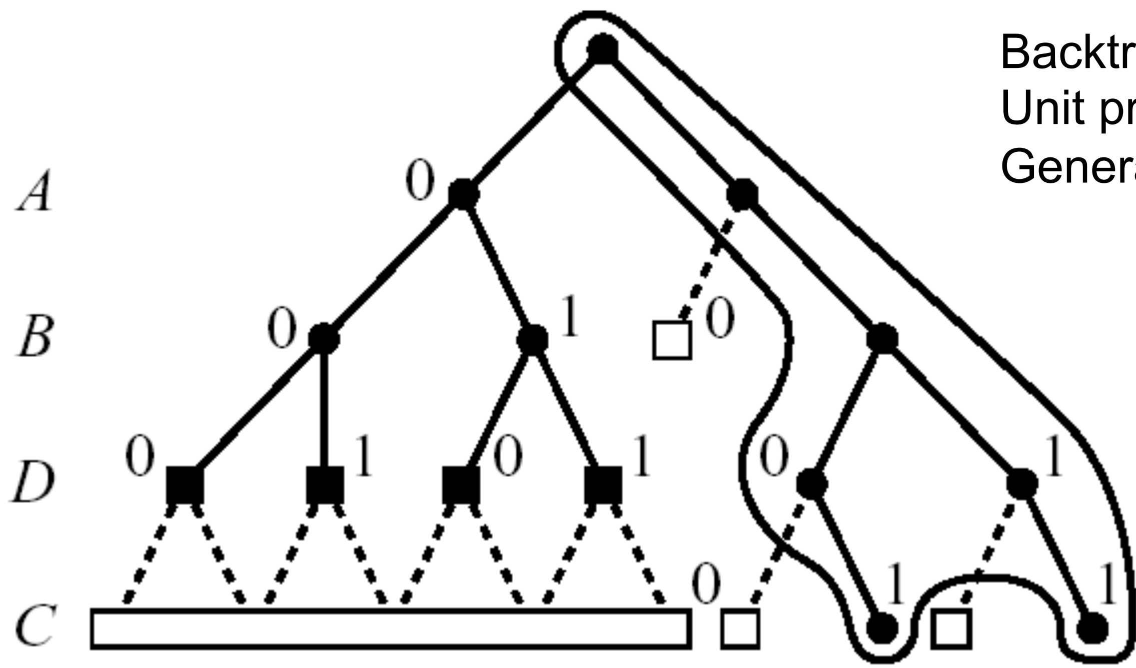
DPLL(φ)
Input: A cnf theory φ
Output: A decision of whether φ is satisfiable.
1. Unit_propagate(φ);
2. If the empty clause is generated, return(*false*);
3. Else, if all variables are assigned, return(*true*);
4. Else
5. Q = some unassigned variable;
6. return(**DPLL**($\varphi \wedge Q$) $\vee$
   DPLL($\varphi \wedge \neg Q$))

Figure 5.13: The DPLL Procedure

Look-ahead for SAT: DPLL

example: (~AVB)(~CVA)(AVBVD)(C)

(Davis-Putnam, Logeman and Laveland, 1962)



Backtracking look-ahead with
Unit propagation=
Generalized arc-consistency

Only enclosed area will be explored with unit-propagation

Look-back:
Backjumping / Learning

- **Backjumping:**
  - In deadends, go back to the most recent culprit.
- **Learning:**
  - constraint-recording, no-good recording.
  - good-recording

Backjumping

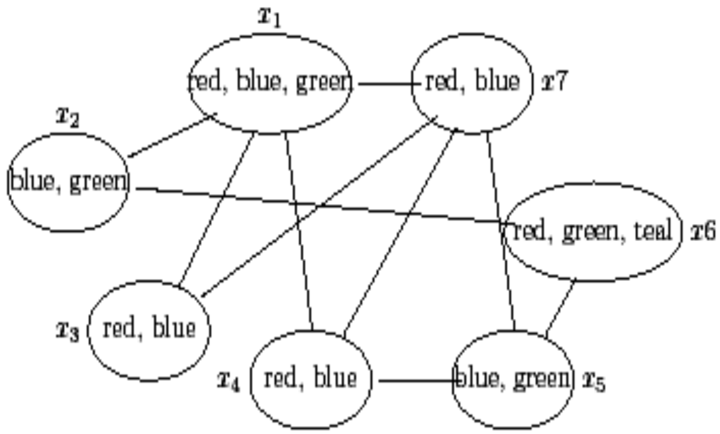
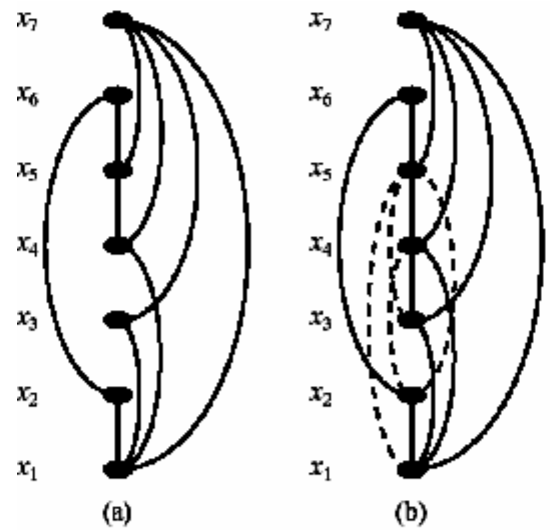


Figure 6.1: A modified coloring problem.



- **(X1=r,x2=b,x3=b,x4=b,x5=g,x6=r,x7={r,b})**
- **(r,b,b,b,g,r)** conflict set **of x7**
- **(r,-,b,b,g,-) c.s. of x7**
- **(r,-,b,-,-,-,-)** minimal conflict-set
- Leaf deadend**: (r,b,b,b,g,r)**
- **Every conflict-set is a** no-good

A coloring problem

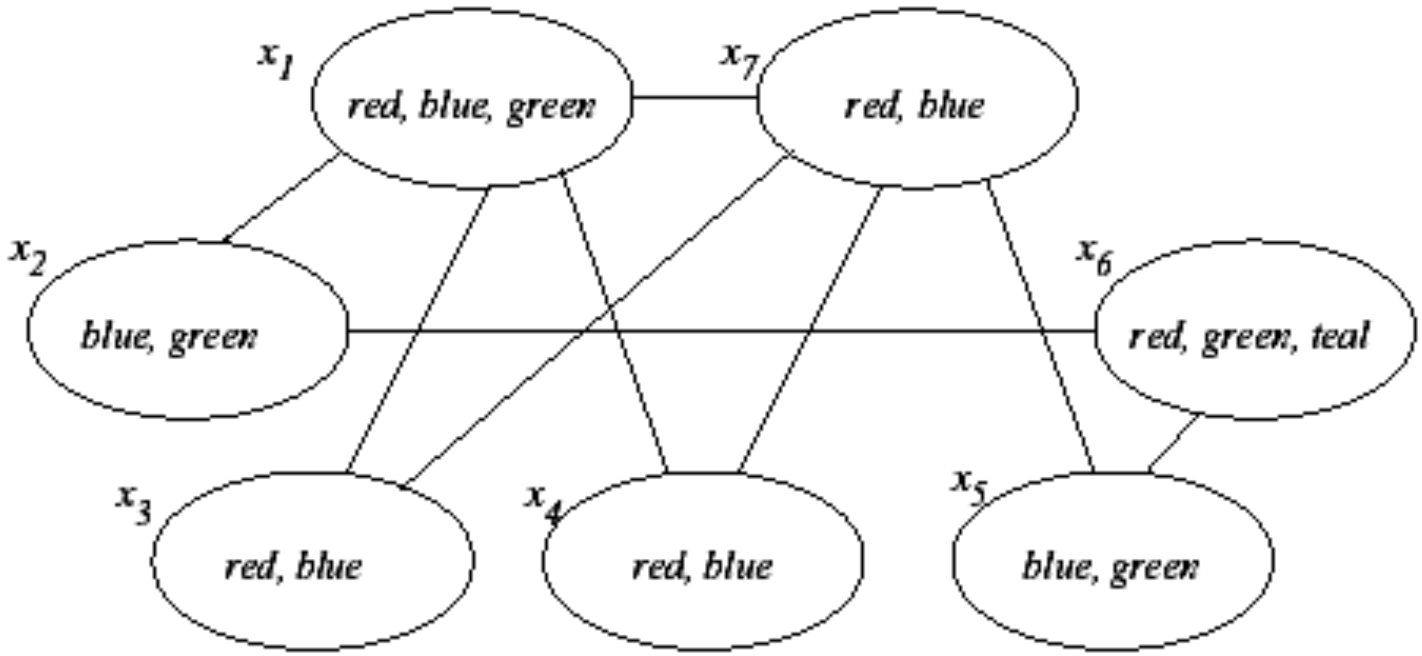


Figure 5.3: A coloring problem with variables $(x_1, x_2, \ldots, x_7)$. The domain of each variable is written inside the corresponding node. Each arc represents the constraint that the two variables it connects must be assigned different colors.

Example of Gaschnig's backjump

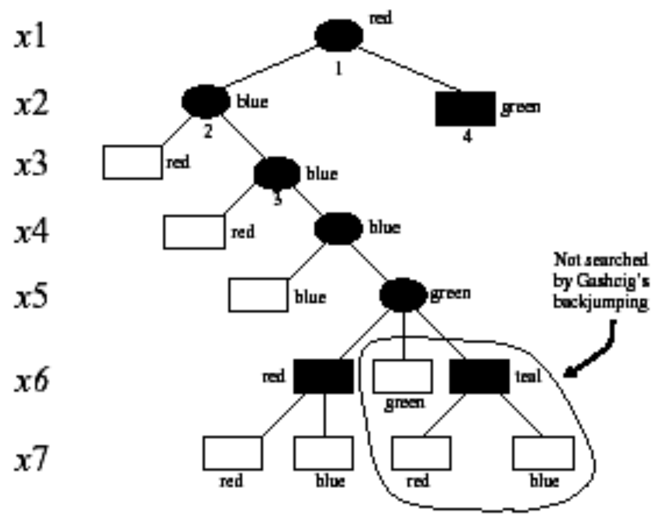
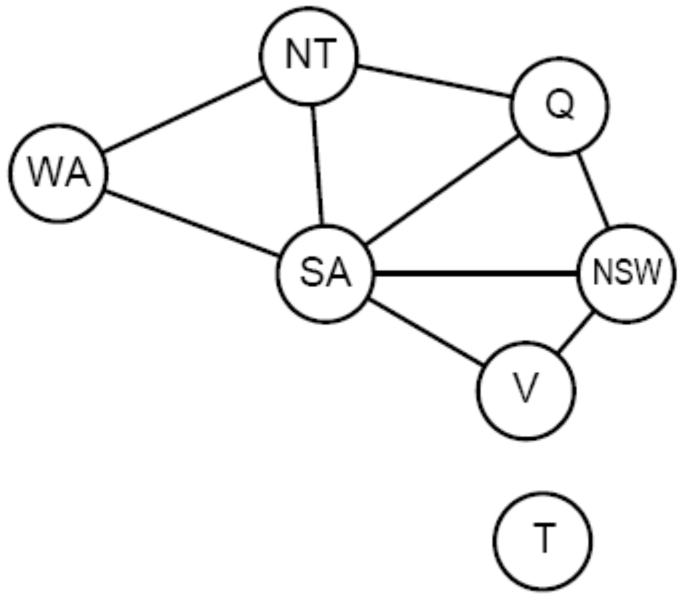


Figure 6.2: Portion of the search space explored by Gaschnig's backjumping, on the example network in Figure 6.1 under $x_1 = red$. The nodes circled are explored by backtracking but not by Gaschnig's backjumping. Notice that unlike previous examples we explicitly display leaf dead-end variables although they are not legal states in the search space.

Example 6.2.3 Consider the problem in Figure 6.1 and the order d_1. At the dead-end for x_7 that results from the partial instantiation $(< x_1, red >, < x_2, blue >, < x_3, blue >, < x_4, blue >, < x_5, green >, < x_6, red >)$, $latest_7 = 3$, because $x_7 = red$ was ruled out by $< x_1, red >$, $x_7 = blue$ was ruled out by $< x_3, blue >$, and no later variable had to be examined. On returning to x_3, the algorithm finds no further values to try ($D'_3 = \emptyset$). Since $latest_3 = 2$, the next variable examined will be x_2. Thus we see the algorithm's ability to backjump at leaf dead-ends. On subsequent dead-ends, as in x_3, it goes back to its preceding variable only. An example of the algorithm's practice of pruning the search space is given in Figure 6.2. □

Problem structure



Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph

Problem structure contd.

Suppose each subproblem has c variables out of n total

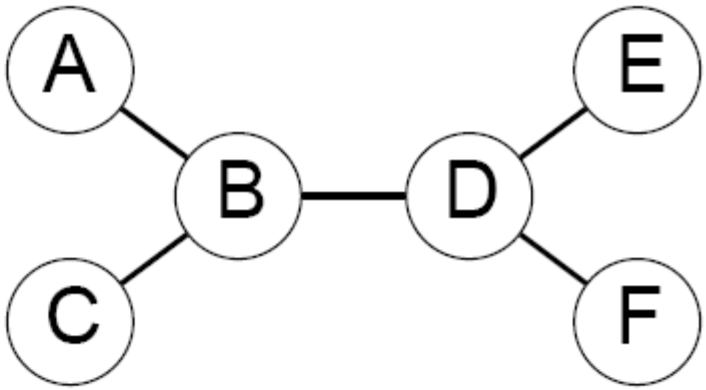
Worst-case solution cost is $n/c \cdot d^c$, *linear* in n

E.g., $n = 80$, $d = 2$, $c = 20$

2^{80} = 4 billion years at 10 million nodes/sec

$4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec

Tree-structured CSPs



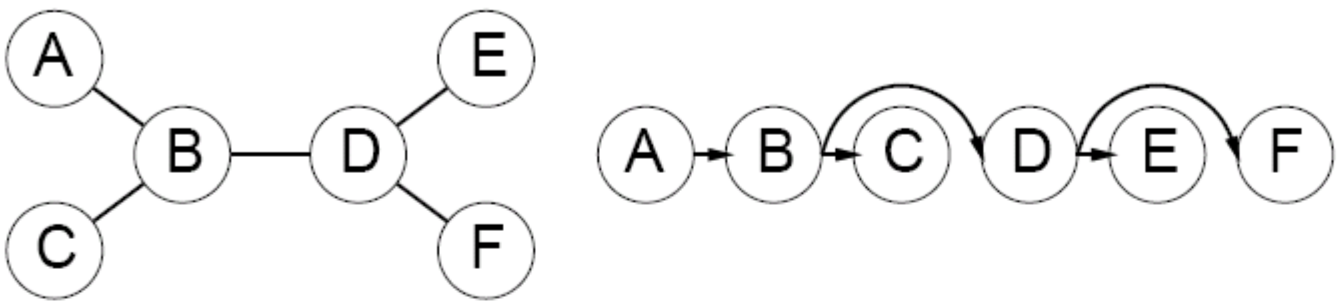
Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time

Compare to general CSPs, where worst-case time is $O(d^n)$

This property also applies to logical and probabilistic reasoning:
an important example of the relation between syntactic restrictions
and the complexity of reasoning.

Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering

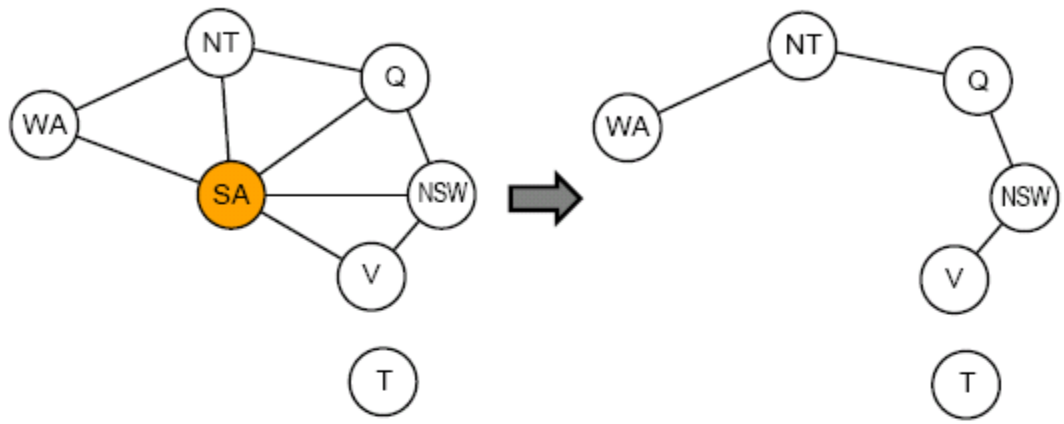


2. For j from n down to 2, apply $\textsc{RemoveInconsistent}(Parent(X_j), X_j)$

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Nearly tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning: instantiate (in all ways) a set of variables
such that the remaining constraint graph is a tree

Cutset size $c \Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c

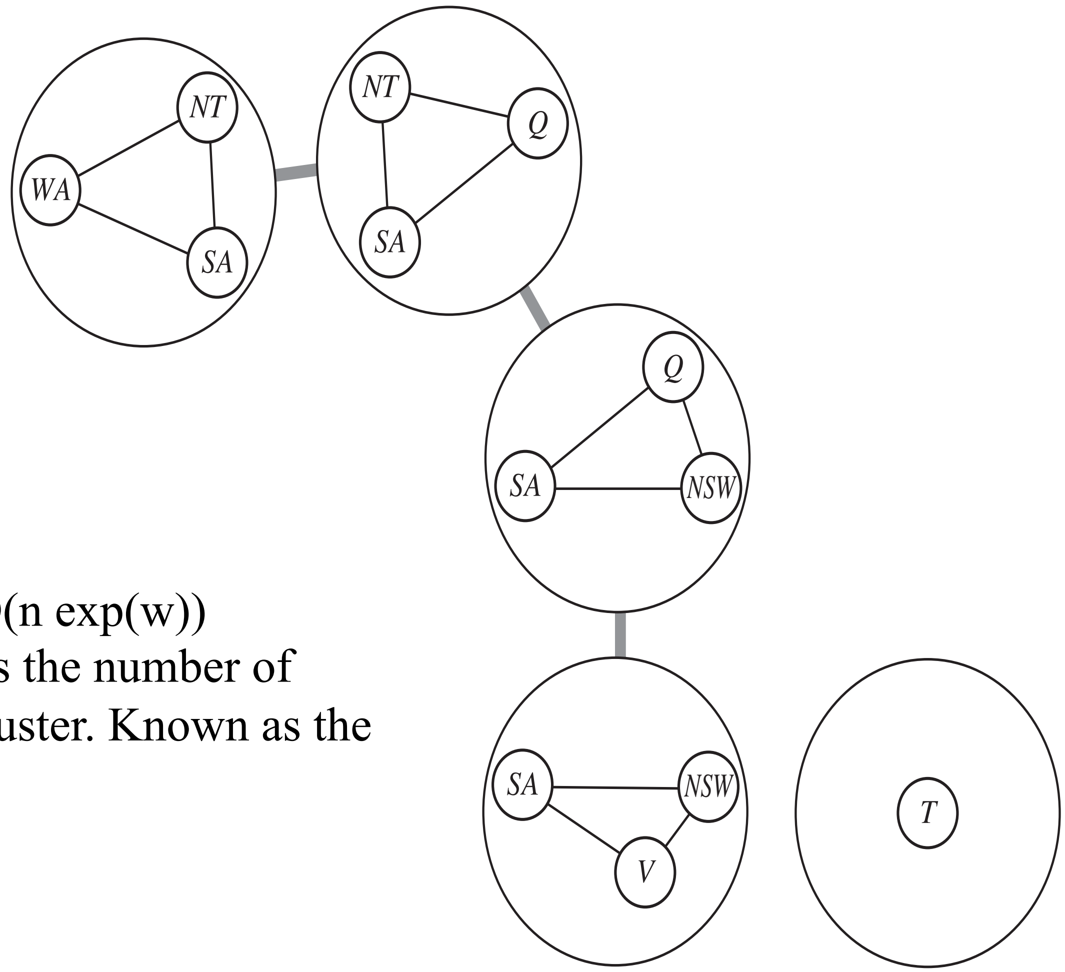
The cycle-cutset method

- **An instantiation can be viewed as blocking cycles in the graph**
- **Given an instantiation to a set of variables that cut all cycles (a cycle-cutset) the rest of the problem can be solved in linear time by a tree algorithm.**

- **Complexity (n number of variables, k the domain size and C the cycle-cutset size):**

$$O(nk^C k^2)$$

Tree Decomposition



Complexity is O(n exp(w))
Where w bounds the number of
Variables in a cluster. Known as the
treewidth

Iterative algorithms for CSPs

Hill-climbing, simulated annealing typically work with
"complete" states, i.e., all variables assigned

To apply to CSPs:
allow states with unsatisfied constraints
operators *reassign* variable values

Variable selection: randomly select any conflicted variable

Value selection by *min-conflicts* heuristic:
choose value that violates the fewest constraints
i.e., hillclimb with $h(n)$ = total number of violated constraints

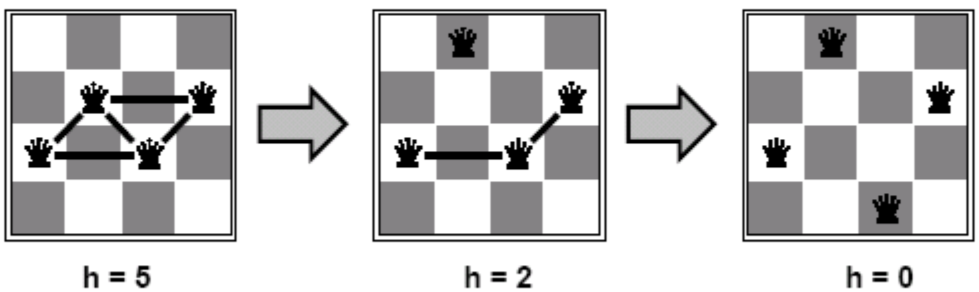
Example: 4-Queens

States: 4 queens in 4 columns ($4^4 = 256$ states)

Operators: move queen in column

Goal test: no attacks

Evaluation: $h(n) =$ number of attacks

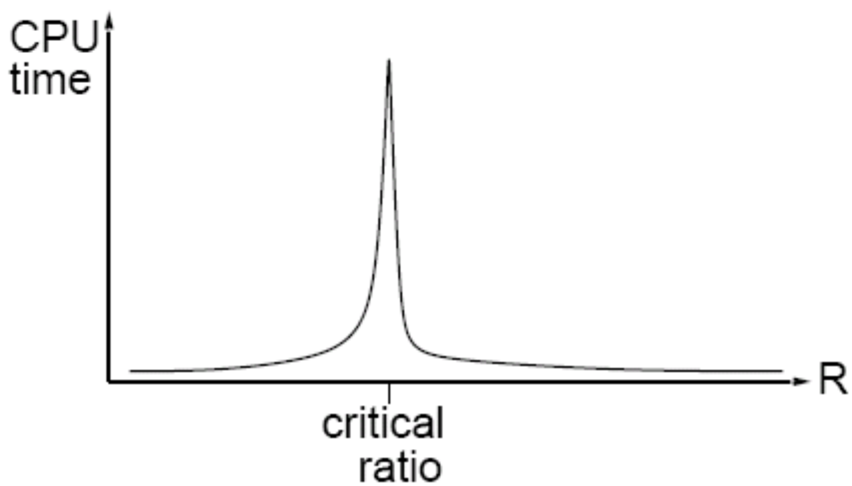


Performance of min-conflicts

Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)

The same appears to be true for any randomly-generated CSP
except in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



GSAT – local search for SAT

(Selman, Levesque and Mitchell, 1992)

```
For i=1 to MaxTries
   Select a random assignment A
   For j=1 to MaxFlips
       if   A satisfies all constraint, return A
       else  flip a variable to maximize the score
            (number of satisfied constraints; if no variable
            assignment increases the score, flip at random)
   end
end
```

Greatly improves hill-climbing by adding
restarts and sideway moves

WalkSAT

(Selman, Kautz and Cohen, 1994)

Adds random walk to GSAT:

With probability p

random walk – flip a variable in some unsatisfied constraint

With probability 1-p

perform a hill-climbing step

Randomized hill-climbing often solves large and hard satisfiable problems

More Stochastic Search: Simulated Annealing, reweighting

- **Simulated annealing:**
  - A method for overcoming local minimas
  - Allows bad moves with some probability:
    - With some probability related to a temperature parameter T the next move is picked randomly.
  - Theoretically, with a slow enough cooling schedule, this algorithm will find the optimal solution. But so will searching randomly.

- **Breakout method (Morris, 1990): adjust the weights of the violated constraints**

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 guarantee later failure

Constraint propagation (e.g., arc consistency) does additional work
to constrain values and detect inconsistencies

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time

Iterative min-conflicts is usually effective in practice