



Artificial Intelligence 1: Constraint Satisfaction problems

Lecturer: Tom Lenaerts

Institut de Recherches Interdisciplinaires et de
Développements en Intelligence Artificielle
(IRIDIA)

Université Libre de Bruxelles



Outline

- CSP?
- Backtracking for CSP
- Local search for CSPs
- Problem structure and decomposition

Constraint satisfaction problems

- What is a CSP?
 - Finite set of variables V_1, V_2, \dots, V_n
 - Finite set of variables C_1, C_2, \dots, C_m
 - Nonempty domain of possible values for each variable $D_{V_1}, D_{V_2}, \dots, D_{V_n}$
 - Each constraint C_i limits the values that variables can take, e.g., $V_1 \neq V_2$
- A *state* is defined as an *assignment* of values to some or all variables.
- *Consistent assignment*: assignment does not not violate the constraints.

October 17, 2004

TLo (IRIDIA)

3

Constraint satisfaction problems

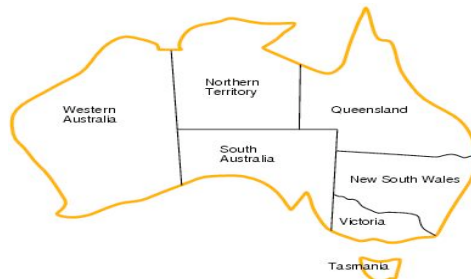
- An assignment is *complete* when every value is mentioned.
- A *solution* to a CSP is a complete assignment that satisfies all constraints.
- Some CSPs require a solution that maximizes an *objective function*.
- Applications: Scheduling the time of observations on the Hubble Space Telescope, Floor planning, Map coloring, Cryptography

October 17, 2004

TLo (IRIDIA)

4

CSP example: map coloring



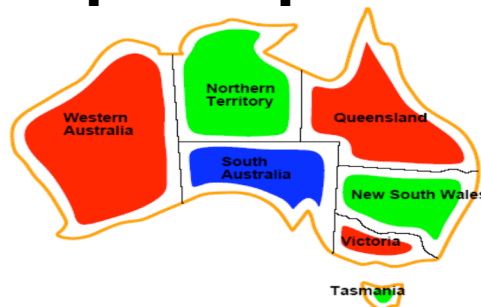
- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D_i = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors.
 - E.g. $WA \neq NT$ (if the language allows this)
 - E.g. $(WA, NT) \neq \{(red, green), (red, blue), (green, red), \dots\}$

October 17, 2004

TLo (IRIDIA)

5

CSP example: map coloring



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

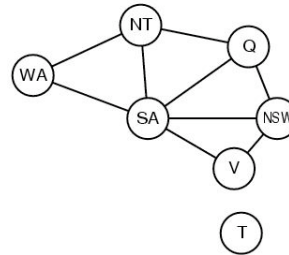
October 17, 2004

TLo (IRIDIA)

6

Constraint graph

- CSP benefits
 - **Standard representation pattern**
 - **Generic goal and successor function**
 - **Generic heuristics (no domain specific expertise).**



- Constraint graph = nodes are variables, edges show constraints.
 - **Graph can be used to simplify search.**
 - e.g. Tasmania is an independent subproblem.

October 17, 2004

TLo (IRIDIA)

7

Varieties of CSPs

- Discrete variables
 - **Finite domains; size $d \Rightarrow O(d^n)$ complete assignments.**
 - E.g. Boolean CSPs, include. 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.**

October 17, 2004

TLo (IRIDIA)

8

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. **cryptarithmic column constraints.**
- Preference (soft constraints) e.g. *red* is better than *green* often representable by a cost for each variable assignment → constrained optimization problems.

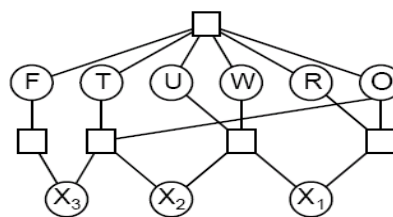
October 17, 2004

TLo (IRIDIA)

9

Example; cryptarithmic

$$\begin{array}{r}
 T W O \\
 + T W O \\
 \hline
 F O U R
 \end{array}$$



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.

October 17, 2004

TLo (IRIDIA)

10

CSP as a standard search problem

- A CSP can easily expressed 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.
 - **Path cost:** as constant cost for every step.

October 17, 2004

TLo (IRIDIA)

11

CSP as a standard search problem

- This is the same for all CSP's !!!
- Solution is found at depth n (if there are n variables).
 - **Hence depth first search can be used.**
- Path is irrelevant, so complete state representation can also be used.
- Branching factor b at the top level is nd .
- $b=(n-l)d$ at depth l , hence $n!d^n$ leaves (only d^n complete assignments).

October 17, 2004

TLo (IRIDIA)

12

Commutativity

- CSPs 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]
 - All CSP search algorithms consider a single variable assignment at a time \Rightarrow there are d^n leaves.

Backtracking search

- Cfr. Depth-first search
- Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Uninformed algorithm
 - **No good general performance (see table p. 143)**

Backtracking search

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

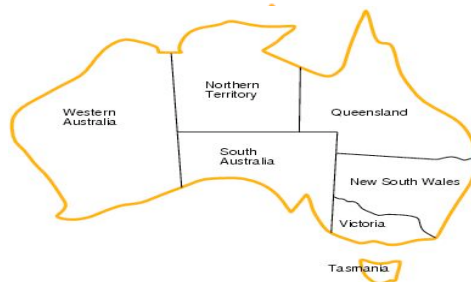
function RECURSIVE-BACKTRACKING(*assignment*, *csp*) **return** a solution or failure
if *assignment* is complete **then return** *assignment*
var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
 if *value*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

October 17, 2004

TLo (IRIDIA)

15

Backtracking example

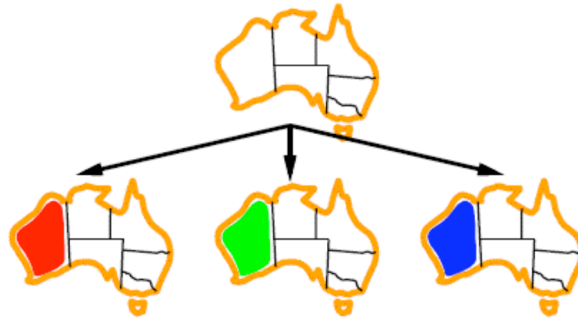


October 17, 2004

TLo (IRIDIA)

16

Backtracking example

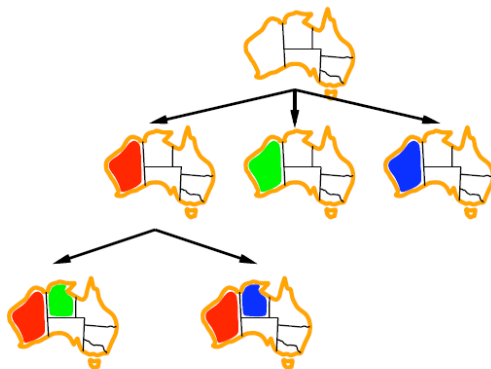


October 17, 2004

TLo (IRIDIA)

17

Backtracking example

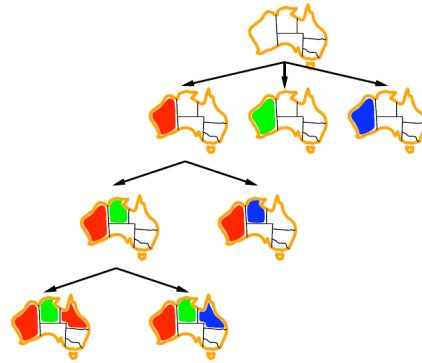


October 17, 2004

TLo (IRIDIA)

18

Backtracking example



October 17, 2004

TLo (IRIDIA)

19

Improving backtracking efficiency

- Previous improvements → introduce heuristics
- 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?
 - Can we take advantage of problem structure?

October 17, 2004

TLo (IRIDIA)

20

Minimum remaining values



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

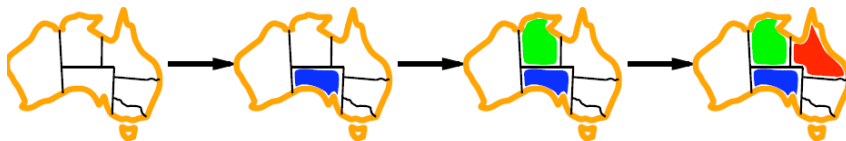
- A.k.a. most constrained variable heuristic
- *Rule*: choose variable with the fewest legal moves
- *Which variable shall we try first?*

October 17, 2004

TLo (IRIDIA)

21

Degree heuristic



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

October 17, 2004

TLo (IRIDIA)

22

Least constraining value



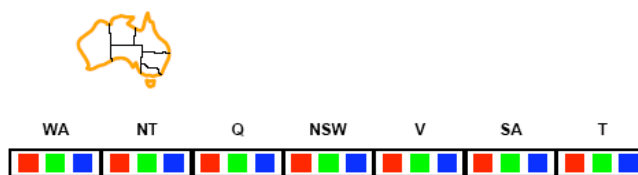
- Least constraining value heuristic
- Rule: given a variable choose the least constraining value i.e. the one that leaves the maximum flexibility for subsequent variable assignments.

October 17, 2004

TLo (IRIDIA)

23

Forward checking



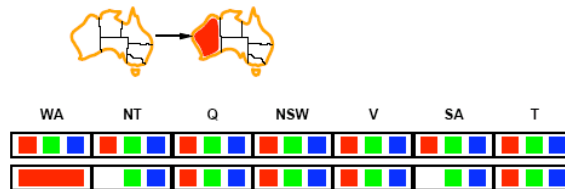
- Can we detect inevitable failure early?
 - **And avoid it later?**
- *Forward checking idea:* keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values.

October 17, 2004

TLo (IRIDIA)

24

Forward checking



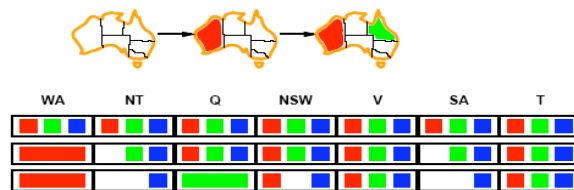
- Assign $\{WA=red\}$
- Effects on other variables connected by constraints with WA
 - **NT can no longer be red**
 - **SA can no longer be red**

October 17, 2004

TLo (IRIDIA)

25

Forward checking



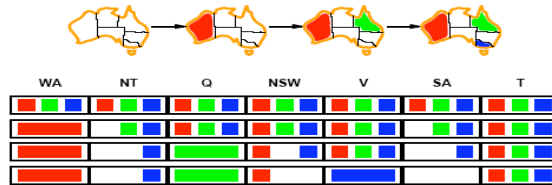
- Assign $\{Q=green\}$
- Effects on other variables connected by constraints with WA
 - **NT can no longer be green**
 - **NSW can no longer be green**
 - **SA can no longer be green**
- *MRV heuristic* will automatically select NT and SA next, why?

October 17, 2004

TLo (IRIDIA)

26

Forward checking



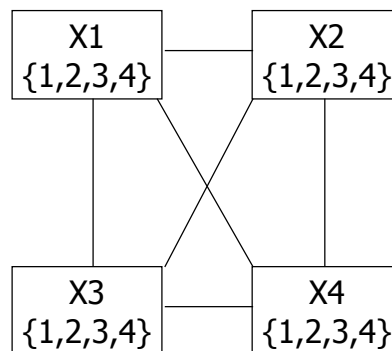
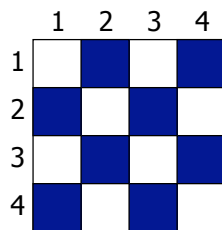
- If V is assigned *blue*
- Effects on other variables connected by constraints with WA
 - **SA is empty**
 - **NSW can no longer be blue**
- FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.

October 17, 2004

TLo (IRIDIA)

27

Example: 4-Queens Problem



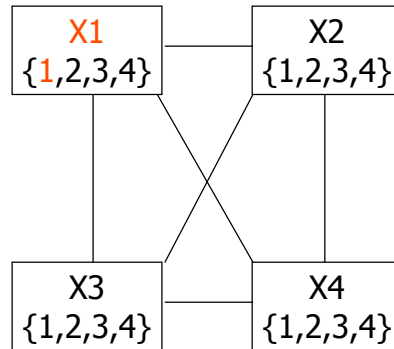
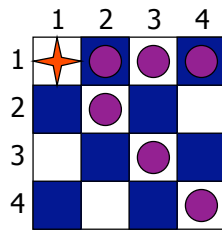
[4-Queens slides copied from B.J. Dorr CMSC 421 course on AI]

October 17, 2004

TLo (IRIDIA)

28

Example: 4-Queens Problem

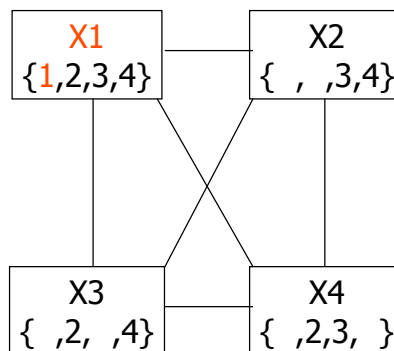
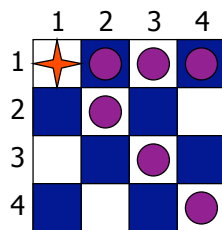


October 17, 2004

TLo (IRIDIA)

29

Example: 4-Queens Problem

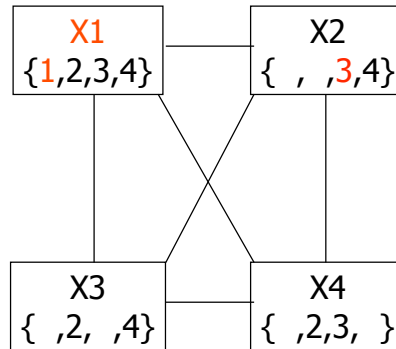
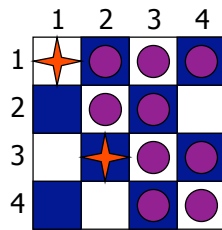


October 17, 2004

TLo (IRIDIA)

30

Example: 4-Queens Problem

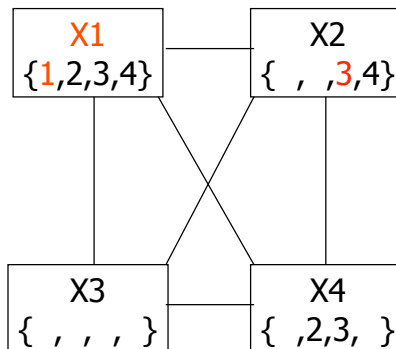
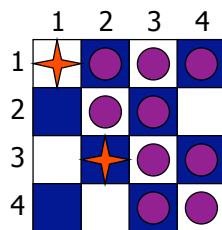


October 17, 2004

TLo (IRIDIA)

31

Example: 4-Queens Problem

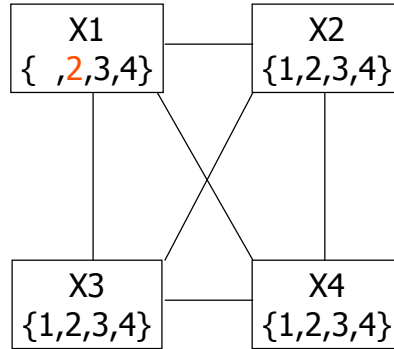
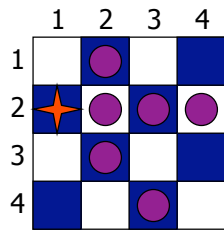


October 17, 2004

TLo (IRIDIA)

32

Example: 4-Queens Problem

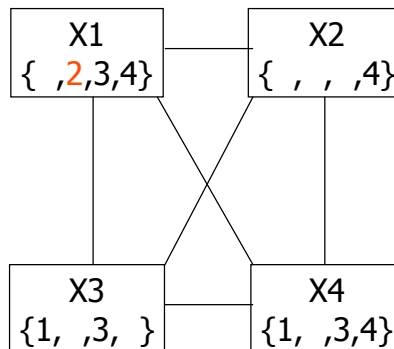
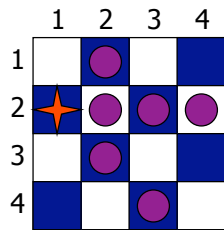


October 17, 2004

TLo (IRIDIA)

33

Example: 4-Queens Problem

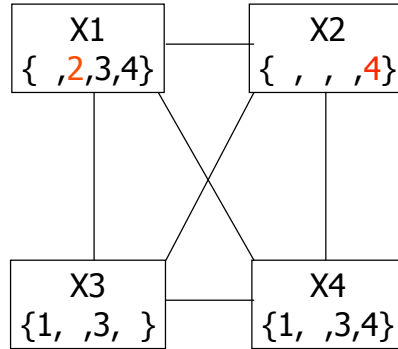
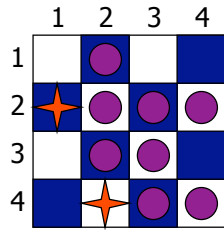


October 17, 2004

TLo (IRIDIA)

34

Example: 4-Queens Problem

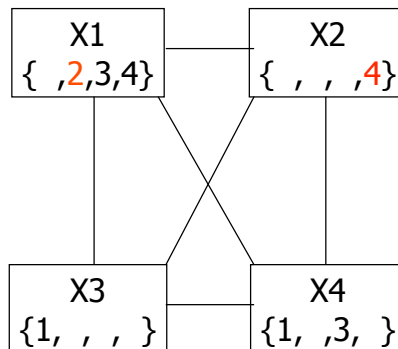
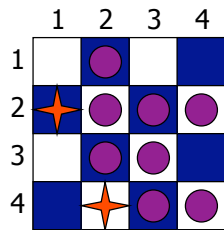


October 17, 2004

TLo (IRIDIA)

35

Example: 4-Queens Problem

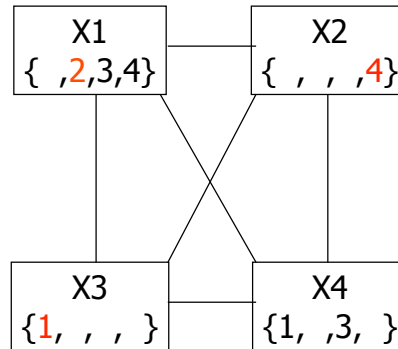
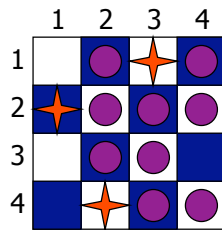


October 17, 2004

TLo (IRIDIA)

36

Example: 4-Queens Problem

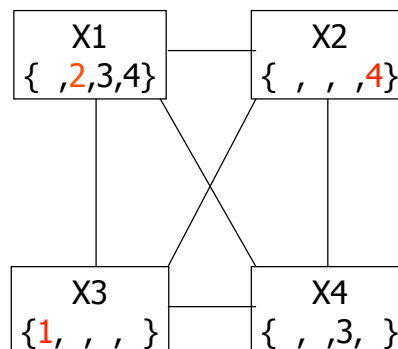
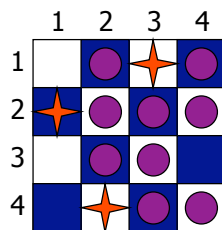


October 17, 2004

TLo (IRIDIA)

37

Example: 4-Queens Problem

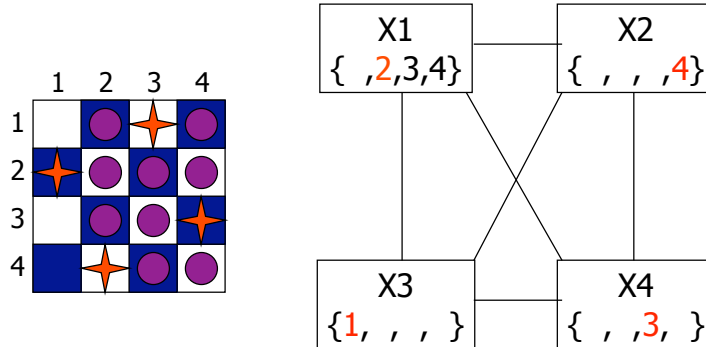


October 17, 2004

TLo (IRIDIA)

38

Example: 4-Queens Problem

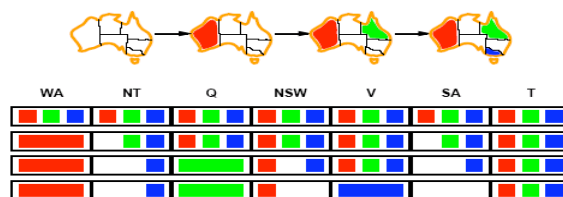


October 17, 2004

TLo (IRIDIA)

39

Constraint propagation



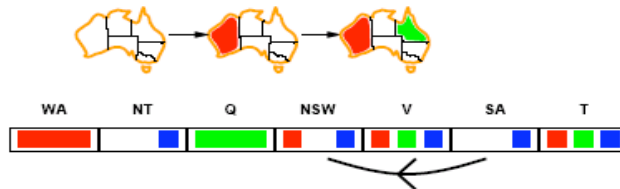
- Solving CSPs with combination of heuristics plus forward checking is more efficient than either approach alone.
- FC checking propagates information from assigned to unassigned variables but does not provide detection for all failures.
 - **NT and SA cannot be blue!**
- Constraint propagation repeatedly enforces constraints locally

October 17, 2004

TLo (IRIDIA)

40

Arc consistency



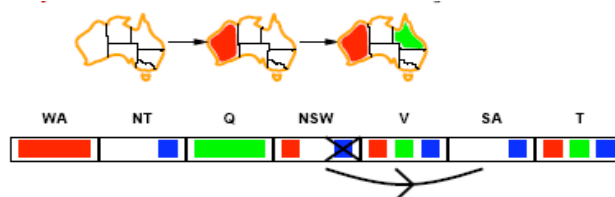
- $X \rightarrow Y$ is consistent iff
for every value x of X there is some allowed y
- $SA \rightarrow NSW$ is consistent iff
 $SA=blue$ and $NSW=red$

October 17, 2004

TLo (IRIDIA)

41

Arc consistency



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

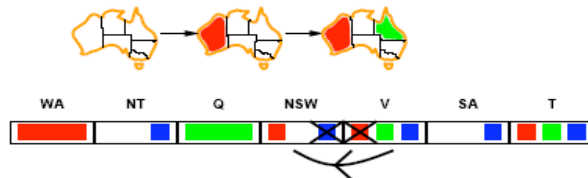
Arc can be made consistent by removing *blue* from *NSW*

October 17, 2004

TLo (IRIDIA)

42

Arc consistency



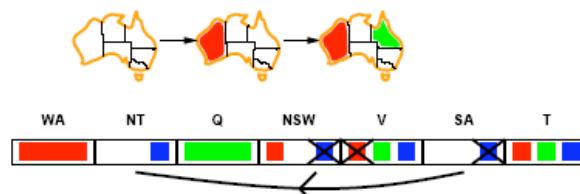
- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbours !!
 - **Remove red from V**

October 17, 2004

TLo (IRIDIA)

43

Arc consistency



- Arc can be made consistent by removing *blue* from *NSW*
- RECHECK neighbours !!
 - **Remove red from V**
- Arc consistency detects failure earlier than FC
- Can be run as a preprocessor or after each assignment.
 - **Repeated until no inconsistency remains**

October 17, 2004

TLo (IRIDIA)

44

Arc consistency algorithm

function AC-3(*csp*) **return** the CSP, possibly with reduced domains

inputs: *csp*, a binary csp with variables $\{X_1, X_2, \dots, X_n\}$

local variables: *queue*, a queue of arcs initially the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$

if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**

for each X_k **in** NEIGHBORS[X_i] **do**

 add (X_i, X_k) to assignment

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*

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.
 - **E.g. 1-consistency or node-consistency**
 - **E.g. 2-consistency or arc-consistency**
 - **E.g. 3-consistency or path-consistency**

K-consistency

- A graph 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.**
- This is ideal since a solution can be found in time $O(nd)$ instead of $O(n^2d^3)$
- YET *no free lunch*: any algorithm for establishing n-consistency must take time exponential in n, in the worst case.

Further improvements

- Checking special constraints
 - **Checking Alldif(...) constraint**
 - *E.g. {WA=red, NSW=red}*
 - **Checking Atmost(...) constraint**
 - *Bounds propagation for larger value domains*
- Intelligent backtracking
 - **Standard form is chronological backtracking i.e. try different value for preceding variable.**
 - **More intelligent, backtrack to conflict set.**
 - Set of variables that caused the failure or set of previously assigned variables that are connected to X by constraints.
 - Backjumping moves back to most recent element of the conflict set.
 - Forward checking can be used to determine conflict set.

Local search for CSP

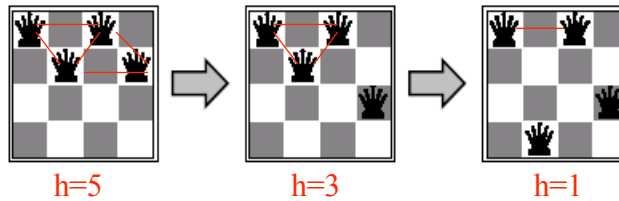
- Use complete-state representation
- For CSPs
 - **allow states with unsatisfied constraints**
 - **operators reassign variable values**
- Variable selection: randomly select any conflicted variable
- Value selection: *min-conflicts heuristic*
 - **Select new value that results in a minimum number of conflicts with the other variables**

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 from VARIABLES[csp]
  value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
  set var = value in current
return failure
```

Min-conflicts example 1



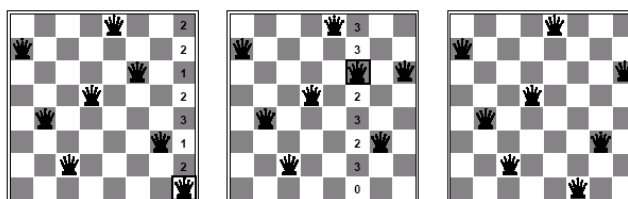
- Use of min-conflicts heuristic in hill-climbing.

October 17, 2004

TLo (IRIDIA)

51

Min-conflicts example 2



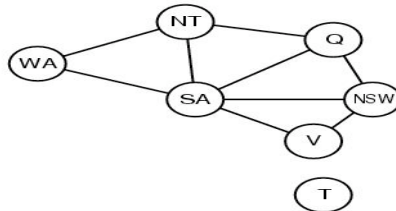
- A two-step solution for an 8-queens problem using min-conflicts heuristic.
- At each stage a queen is chosen for reassignment in its column.
- The algorithm moves the queen to the min-conflict square breaking ties randomly.

October 17, 2004

TLo (IRIDIA)

52

Problem structure



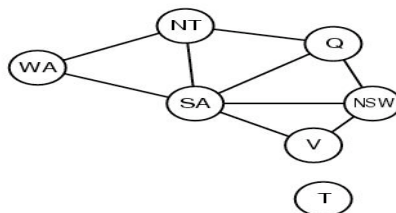
- How can the problem structure help to find a solution quickly?
- Subproblem identification is important:
 - Coloring Tasmania and mainland are independent subproblems
 - Identifiable as connected components of constrained graph.
- Improves performance

October 17, 2004

TLo (IRIDIA)

53

Problem structure



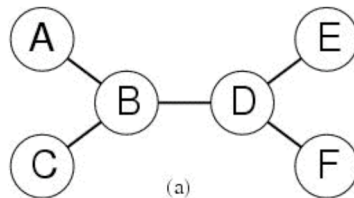
- Suppose each problem has c variables out of a total of n .
- Worst case solution cost is $O(n/c d^c)$, i.e. linear in n
 - Instead of $O(d^n)$, exponential in n
- E.g. $n=80, c=20, d=2$
 - $2^{80} = 4$ billion years at 1 million nodes/sec.
 - $4 * 2^{20} = .4$ second at 1 million nodes/sec

October 17, 2004

TLo (IRIDIA)

54

Tree-structured CSPs



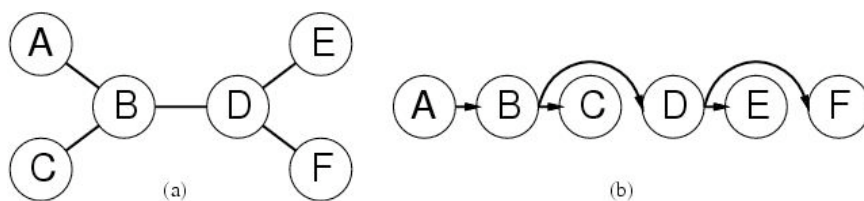
- Theorem: if the constraint graph has no loops then CSP can be solved in $O(nd^2)$ time
- Compare difference with general CSP, where worst case is $O(d^n)$

October 17, 2004

TLo (IRIDIA)

55

Tree-structured CSPs



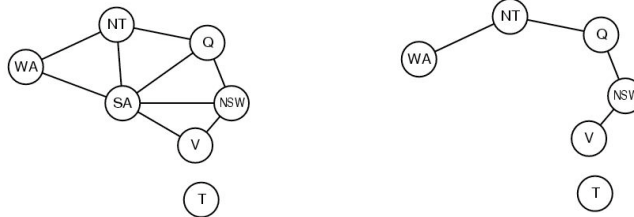
- In most cases subproblems of a CSP are connected as a tree
- Any tree-structured CSP can be solved in time linear in the number of variables.
 - Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering.
 - For j from n down to 2, apply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

October 17, 2004

TLo (IRIDIA)

56

Nearly tree-structured CSPs



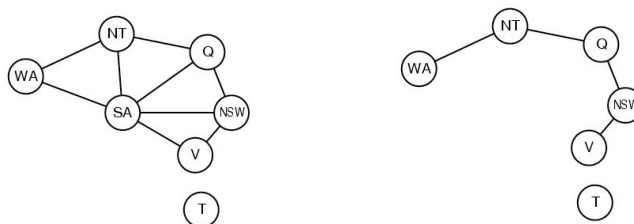
- Can more general constraint graphs be reduced to trees?
- Two approaches:
 - Remove certain nodes
 - Collapse certain nodes

October 17, 2004

TLo (IRIDIA)

57

Nearly tree-structured CSPs



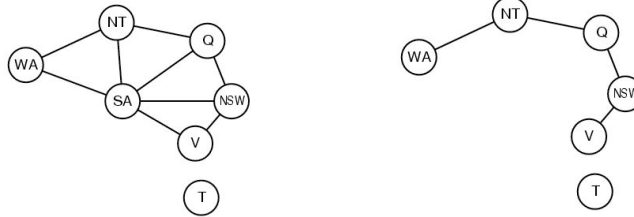
- Idea: assign values to some variables so that the remaining variables form a tree.
- Assume that we assign $\{SA=x\} \leftarrow \text{cycle cutset}$
 - And remove any values from the other variables that are inconsistent.
 - The selected value for SA could be the wrong one so we have to try all of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

October 17, 2004

TLo (IRIDIA)

58

Nearly tree-structured CSPs



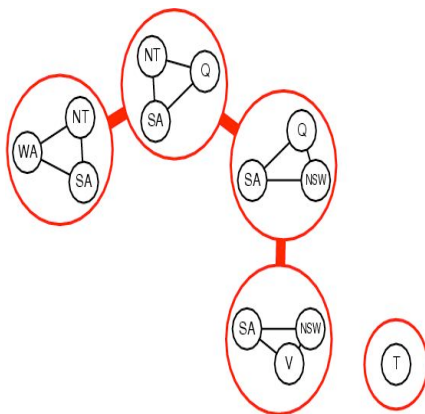
- This approach is worthwhile if cycle cutset is small.
- Finding the smallest cycle cutset is NP-hard
 - **Approximation algorithms exist**
- This approach is called *cutset conditioning*.

October 17, 2004

TLo (IRIDIA)

59

Nearly tree-structured CSPs



- Tree decomposition of the constraint graph in a set of connected subproblems.
- Each subproblem is solved independently
- Resulting solutions are combined.
- Necessary requirements:
 - **Every variable appears in at least one of the subproblems.**
 - **If two variables are connected in the original problem, they must appear together in at least one subproblem.**
 - **If a variable appears in two subproblems, it must appear in each node on the path.**

October 17, 2004

TLo (IRIDIA)

60

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.
- 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.