TDT4136 Introduction to Artificial Intelligence

Lecture 6 - Constraint Satisfaction Problems

Chapter 5 in textbook

Pinar Özturk

Norwegian University of Science and Technology

2021

Outline

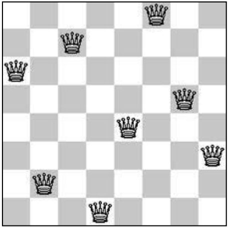
What is a constraint satisfaction problem?

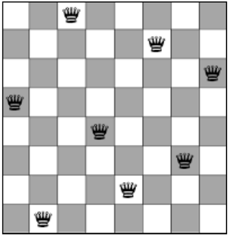
Example: N-queen problem

Place N queens on an NxN chess board with the constraint that they don't attack each other

8-queen problem

Solution to the 8-queen problem?





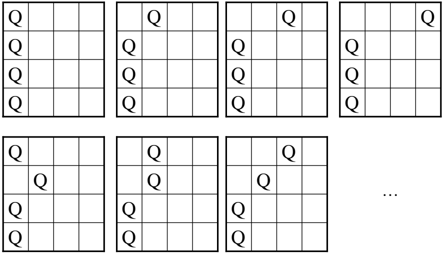
How would you solve the N-queen problem?

Humans solve this problem

- by experimenting with different configurations.
- using various insights about the problem to explore only a small number of configurations before they find an answer.
- However, it would be hard for humans to solve a 1000 Queen problem!

We need computer-specific methods for solving such a problem.

Method: Try every configuration systemically?

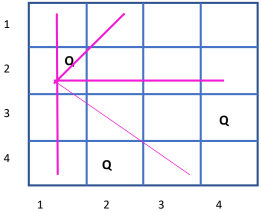


Although computers are good at trying many small things quickly

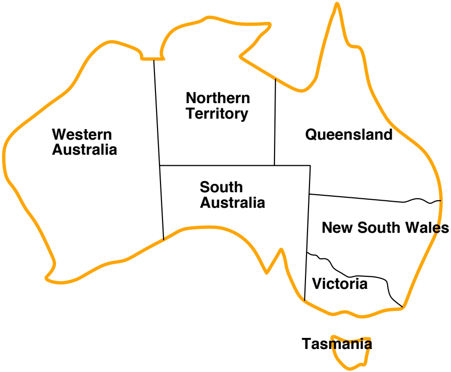
- for the 4-queen problem there are 256 configurations
- 16,777,216 configurations for 8-queen problem, and
- ...
- there are N^{N} configurations for N-Queens.
- still too much time for the modern machines

Good news

- We notice that in, e.g., the 4-Queens problem, as soon as we place some of the queens we know that an entire additional set of configurations are invalid.
- This is the rationale behind the methods for solving CS problems.
- N-queen problem is an instance of a generic problem class and we want to design algorithms that solve this type of problems.



Example: Map-Coloring



Task/problem: colour this map using 3 colours with the **constraint:** adjacent regions must have different colours

CSP in real-world

Assignment problems

e.g., who teaches what class

Timetabling problems

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

Hardware configuration

Transportation scheduling

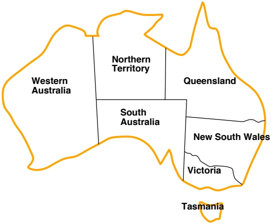
Factory scheduling

etc.

CSP definition, components

- A Constraint Satisfaction Problem (CSP) is defined through 3 components:
  - A set of **variables**
  - A set of **values**
  - A set of **constraints** between variables
- A CSP solving algorithm should assign a value to each variable that satisfies all the constraints .

Example: Map-Coloring



Variables: $\{WA, NT, Q, NSW, V, SA, T\}$
Domain: {red,green,blue}
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

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,

e.g., Sudoku

Preferences (soft constraints), e.g., *red* is better than *green*
often representable by a cost for each variable assignment

$\rightarrow$ constrained optimization problems

Crytoarithmetic problem with higher-order constraints

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

- **Variables:**

$F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

- **Domains:**

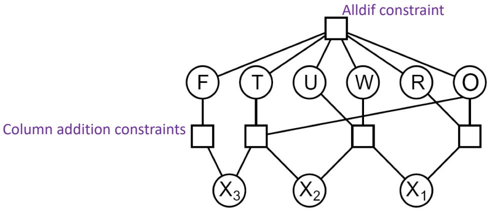
$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- **Constraints:**

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

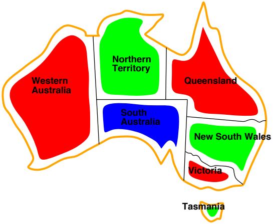
$O + O = R + 10 \cdot X_1$

$\cdots$



Cryptaritmetic problem:
Variables X1, X2, X3 shows carry digits
Boxes show constraints

What is a Solution?

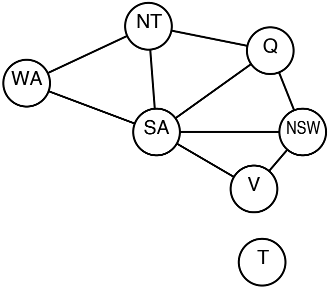


Solutions are assignments satisfying all constraints, e.g.,

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

Visualization of CSP - Constraint graph

Constraint graph: nodes are variables, arcs show constraints, e.g, in the fllowing figure: WA and NT cannot take the same value.



Basic problem: Find a $d_i \in D$ for each variable V_i such that all constraints are satisfied, i.e, find consistent values for variables

Approaches to Solving CSPs

- Inference - Constraint propagation
- Incremental Search - Backtracking
- Search with inference, e..g., Backtracking with Forward Checking
- Local Search

Inference for CSP

- A specific type of inference called **constraint propagation**
  - node consistency, arc consistency, path consistency, K-consistency, forward checking (restricted arc consistency)
- Objective: reduce legal values for a variable using constraints
- Reduction in legal values of a variable in turn reduces legal values of another variable....

Node consistency

- related to unary constraints for a variable
- e.g., $SA \neq green$ in the map colouring problem
- achieved by eliminating values from the domain of the variable

Arc consistency

Arc consistency eliminates the values from the domains of variables that can not be part of a solution.

An arc $V_i \rightarrow V_j$ is consistent if

For every x in D_i , there exist a y in D_j such that (x,y) is allowed by the constraint on the arc.

We can obtain arc consistency by removing values from the domains of the variables that fail the constraint.

Arc consistency algorithm

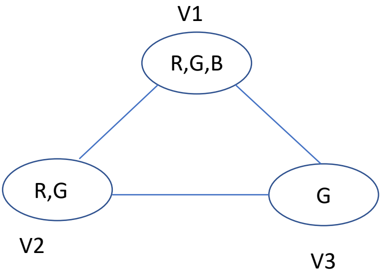
function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  queue ← a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i,\ X_j) \leftarrow$ POP(*queue*)
    if REVISE(*csp*, $X_i,\ X_j$) **then**
      if size of $D_i\ =\ 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to *queue*
  return *true*

function REVISE(*csp*, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  revised ← *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised ← *true*
  return *revised*

Arc Consistency - Example

Three variables, V1, V2, V3 and their initial domains. Values within each node show the initial domains of the variables.



R: Red, G: Green, B: Blue

Constraints:

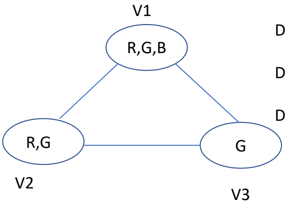
DIFF-color(V1,V3)

DIFF-color(V1,V2)

DIFF-color(V2,V3)

Arc Consistency - Example

- A binary constraint (e.g., DIFF(V1, V2)) can be satisfied through checking 2 arcs, e.g. V1V2 and V2V1.
- All arcs are checked for consisteny, e.g, according to algorithm AC-3



Constraints:

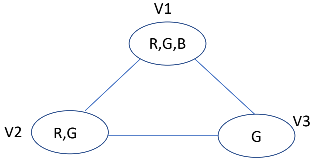
DIFF-color(V1,V3)

DIFF-color(V1,V2)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted
V1-V2	
V2-V1	
V1-V3	
V3-V1	
V2-V3	
V3-V2	

Arc Consistency - Example, cont



Constraints:

DIFF-color(V1,V2)

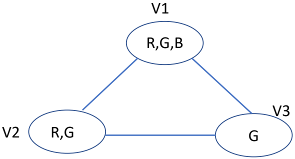
DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3		
V3-V1		
V2-V3		
V3-V2		

- For the arc V1-V2: check if there is a value in the domain of V2 for each value of V1 (i.e. R, G, B)
- For the arc V2-V1: check if there is a value in the domain of V1 for each value of V2 (i.e. R, G)

Arc Consistency - Example, cont



Constraints:

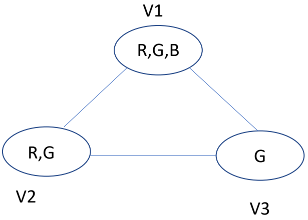
DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3	G	(V1={R,B})
V3-V1		
V2-V3		
V3-V2		
V2-V1		

Arc Consistency - Example, cont



Constraints:

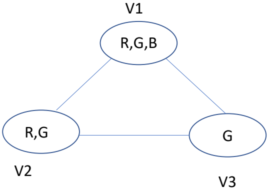
DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3	G	(V1={R,B})
V3-V1	none	(V3={G})
V2-V3	G	(V2={R})
V3-V2		
V2-V1		
V1-V2		

Arc Consistency - Example, cont



Constraints:

DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3	G	(V1={R,B})
V3-V1	none	(V3={G})
V2-V3	G	(V2={R})
V3-V2	none	
V2-V1	none	
V1-V2	R	(V1={B})
V2-V1	none	
V3-V1	none	

We stopped when there is no more changes.

Arc Consistency.

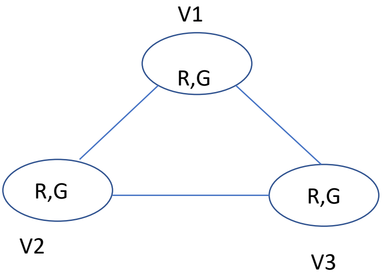
If one of the Domains become empty: there is no solution to the problem

ELSE: Arc consistency is required for existence of a solution - i.e., no empty domains.

BUT: Is Arc consistency sufficient to find a solution?

Arc Consistency - Example, cont.

Is this graph Arc-consistent?
Solution?



Constraints:

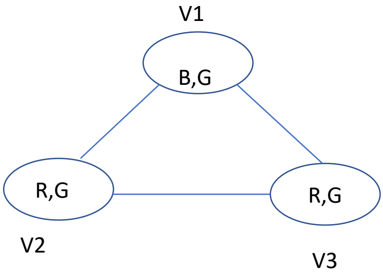
DIFF-color(V1,V3)

DIFF-color(V1,V2)

DIFF-color(V2,V3)

Arc Consistency - Example, cont.

Is this graph Arc-consistent?
Solution?



Constraints:

DIFF-color(V1,V3)

DIFF-color(V1,V2)

DIFF-color(V2,V3)

MESSAGE: SEARCH may be necessary to find a solution.

PS! However, sometimes Arc-consistency alone may find the solution.

CSP as a Standard Search problem

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.
  $\Longrightarrow$ fail if no legal assignments (not fixable!)
- ♢ intermediate states: partial assignment of variables
- ♢ Goal test: the current assignment is complete and consistent

Backtracking search

- Variable assignments are commutative, i.e.,

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

  same as

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

- Backtracking employs **Depth-first search** for CSPs with single-variable assignments
- Backtracking search is the basic uninformed algorithm for CSPs
- Only need to consider assignments to a single variable at each node

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

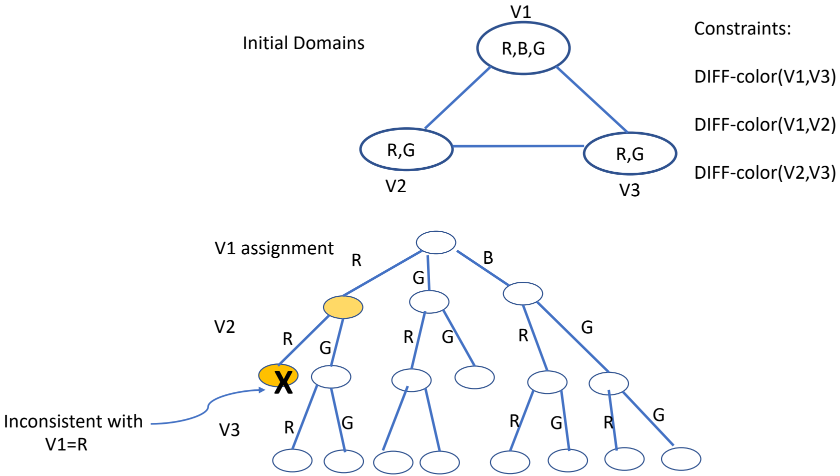
Can solve n-queens for $n \approx 25$

Backtracking search

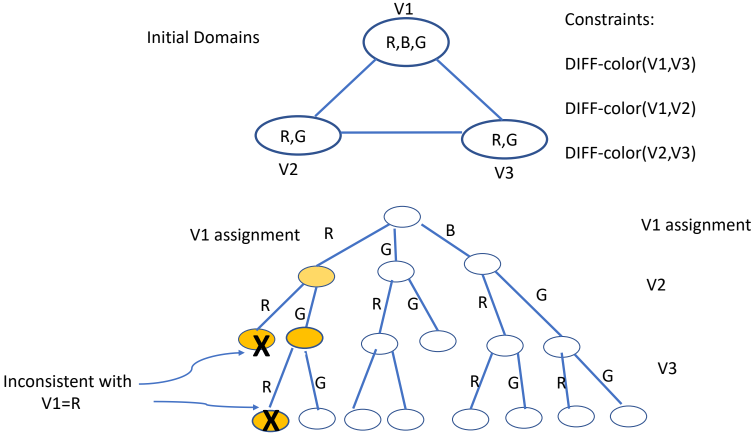
```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
           add inferences to csp
           result ← BACKTRACK(csp, assignment)
           if result ≠ failure then return result
           remove inferences from csp
        remove {var = value} from assignment
  return failure
```

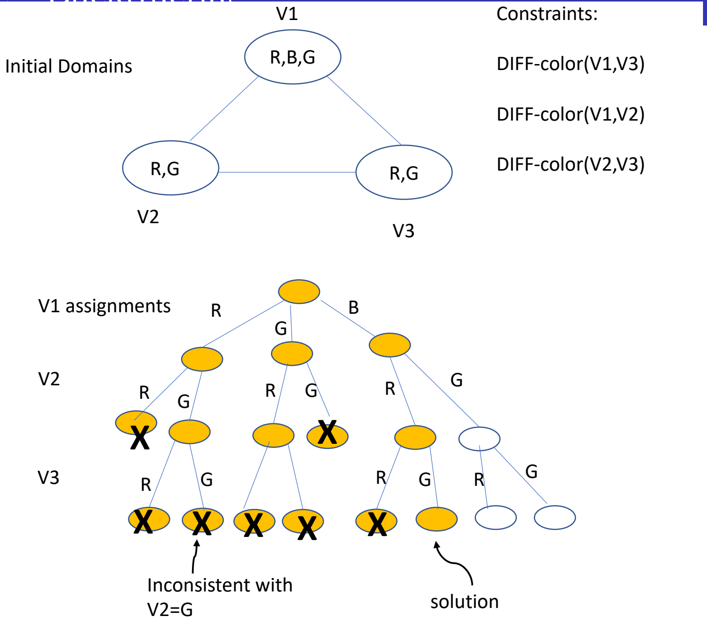
Example - Backtracting



Example - Backtracting



Example - Backtracking



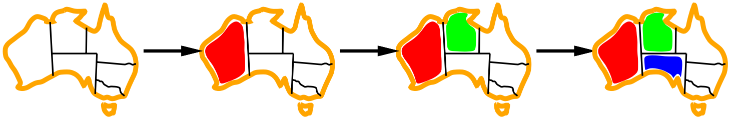
There are two solutions.

Improving backtracking efficiency by Heuristics

- Previously we talked about (domain-specific) heuristics for improvement of uninformed search
- This time heuristics are NOT domain-specific though
- For CSP, *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?

Order of variables

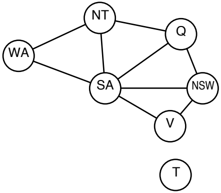
The choice of variable to be assigned next makes a difference.



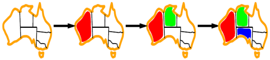
- Assume now this static order: WA, NT, Q, NSW, V, SA, T, and then these 2 assignments: WA = RED, NT = GREEN.
- Q would be default next selection. But is it a good choice .... After WA and NT are assigned? Problem for SA?

Minimum remaining values (MRV) heuristic

Which variable should be assigned next?



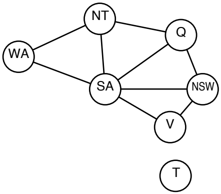
- **Minimum remaining values (MRV) heuristic**: choose the variable with the fewest legal values
- Also called "most constrained variable" or "fail first" heuristic
- Objective: detect immediately if variable has no legal values(left) or most likely to cause a failure soon

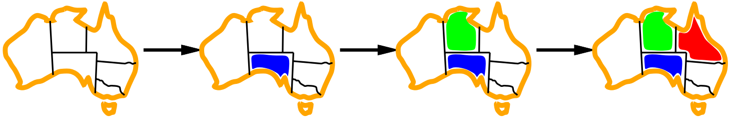


Degree heuristic

Which variable should be assigned next?

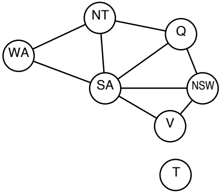
- Does MRV help in the very beginning?
- **Degree heuristic**: choose the variable with the most constraints on remaining variables
- Objective: reduce the future branching on future choices
- Tie-breaker among MRV variables

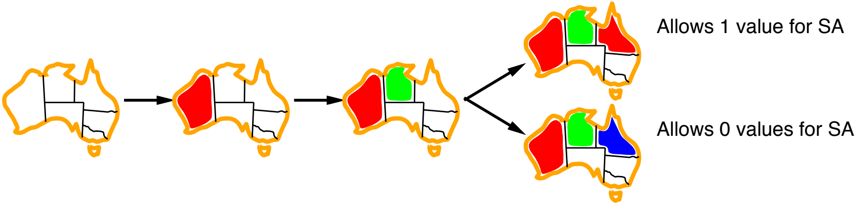




What order should its values be tried?

- Heuristic for decision on value ordering.
- Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables
- Objective: make more likely to find a solution early





Combining these heuristics makes 1000 queens feasible

Improving Backtracking search through Inference

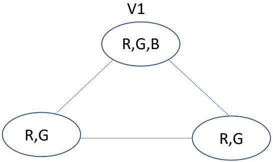
- It is possible to discover the unpromising nodes by inference for consistency
- Arc-consistency can be used during search, however time consuming
- Another possible type of inference is **Forward Checking** of consistency
- Backtracking search can be pruned by applying Forward Checking
- Forward checking:
  - is simpler than Arc consistency, checks only the direct neighbours
  - keep track of remaining legal values for unassigned variables
  - checks "future" rather than "past" which is what bactracking does.

Bactracking with Forward checking

- During search, assume assignment of a value to the next variable and check if it reduces the domain of its neighbours.

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

Example: Backtracking with Forward checking



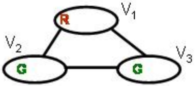
Constraints:

DIFF-color(V1,V2)

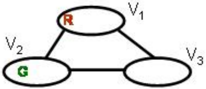
DIFF-color(V1,V3)

DIFF-color(V2,V3)

Variable assign.	Forward check; value deleted	backtrack
V1=R	V2={G}, V3={G}	no

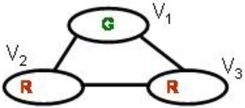


Variable assign.	Forward check; value deleted	backtrack
V1=R	V2={G}, V3={G}	no
V2=G	V3={ }	yes

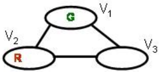


Example: Backtracking with Forward checking - cont.

Variable assign.	Forward check; value deleted	backtrack
V1=R	V2={G}, V3={G}	no
V2=G	V3={ }	yes
V1= G	V2={R}, V3={R}	no

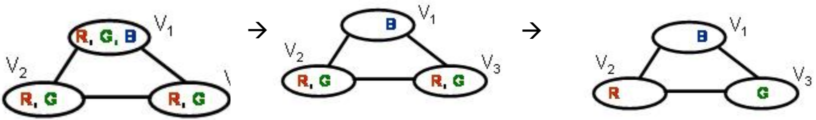


Variable assign.	Forward check; value deleted	backtrack
V1=R	V2={G}, V3={G}	no
V2=G	V3={ }	yes
V1= G	V2={R}, V3={R}	no
V2= R	V3={ }	yes



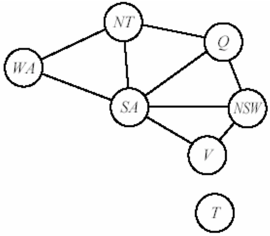
Example: Backtracking with Forward checking - cont.

Variable assign.	Forward check; value deleted	backtrack
V1=R	V2={G}, V3={G}	no
V2=G	V3={ }	yes
V1= G	V2={R}, V3={R}	no
V2= R	V3={ }	yes
V1=B	none	no
V2=R	V3={G }	no



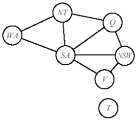
Backtracking Search with Forward checking - Map colouring example

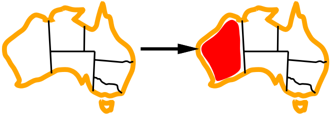
Idea: Keep track of remaining legal values for unassigned variables
Idea: Terminate search when any variable has no legal values



Forward checking - Map colouring example

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

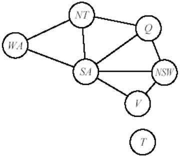


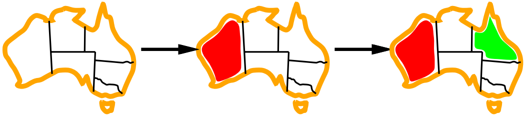


Forward checking - Map colouring example

Idea: Keep track of remaining legal values for unassigned variables

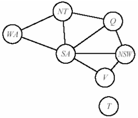
Idea: Terminate search when any variable has no legal values

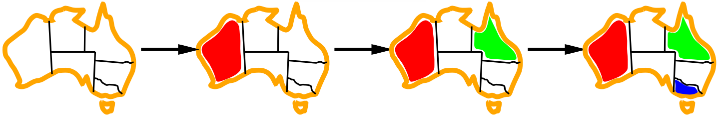


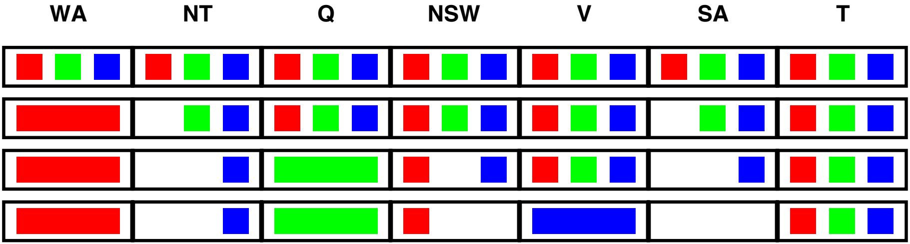


Idea: Keep track of remaining legal values for unassigned variables

Idea: Terminate search when any variable has no legal values

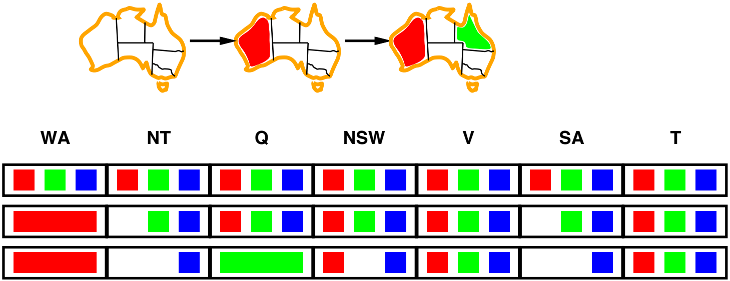






Arc Consistency for Map Coloring

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!

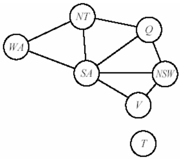
Arc consisteny repeatedly enforces constraints locally

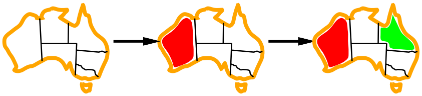
Arc consistency for Map Coloring

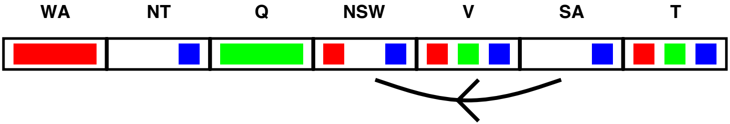
Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y





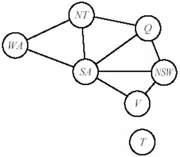


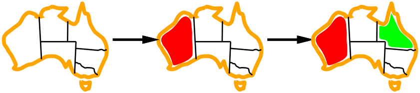
Arc consistency for Map Coloring

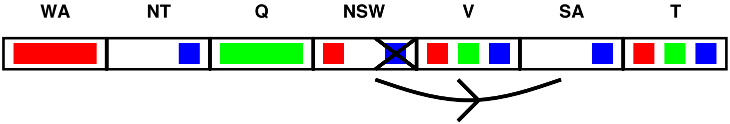
Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y





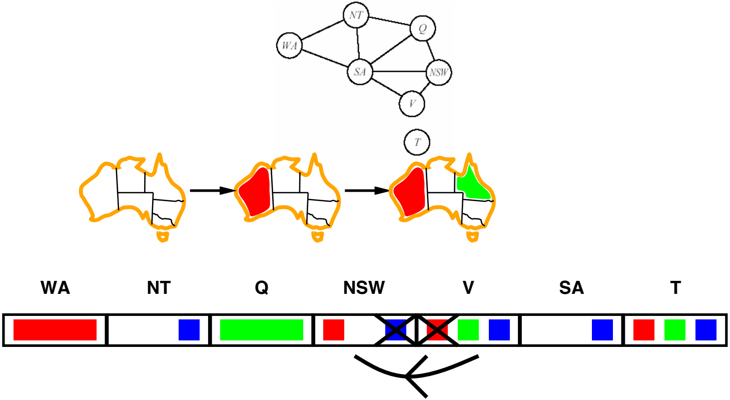


Arc consistency for Map Coloring

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



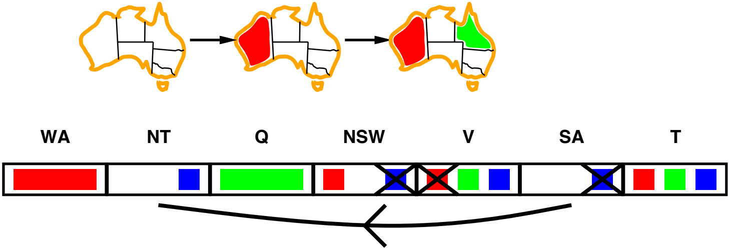
If X loses a value, neighbors of X need to be rechecked

Arc consistency for Map Coloring

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 detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Local search applied to CSP

- Iterative
- Generate complete assignments: assume/guess a value for each variable.
- Evaluate the assignment w.r.t. violated constraints.
- Modify the assignments to reduce the number of violations.

Local search with heuristic

To apply to CSPs:

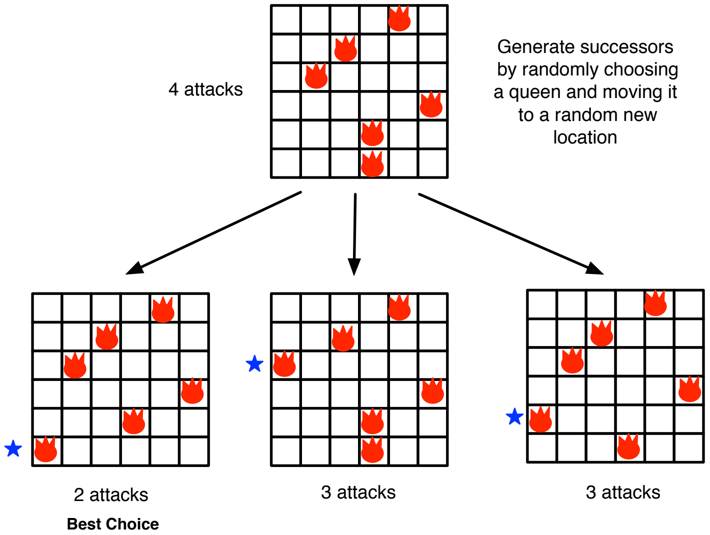
- allow states with unsatisfied constraints
- operators reassign variable values

Variable selection: randomly select any conflicted variable

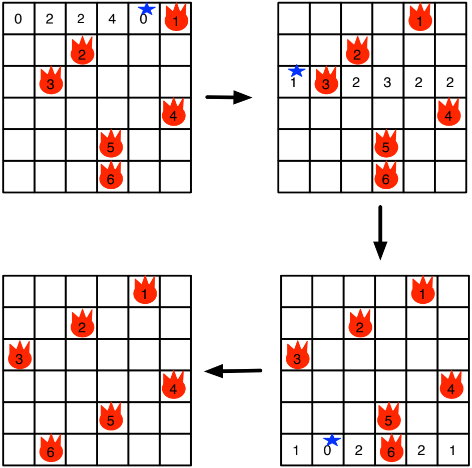
Value selection: by min-conflicts heuristic:
  choose value that violates the fewest constraints
  i.e., hillclimb with $h(n)$ = total number of violated constraints

The local search strategies (e.g., hill-climbing, simulated annealing) in Lecture 4 are candidates for use in CSP.

Solving K-Queens with Dumb Local Search

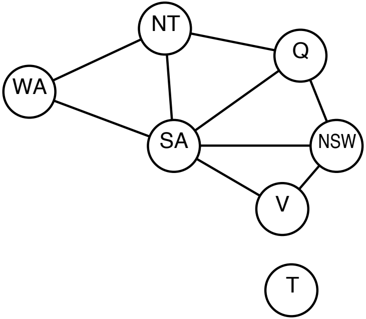


Min Conflicts: Local Search with Intelligence



- Moving queen to column of least violations → intelligent successor generation.
- Integers denote number of violations if queen moved there.

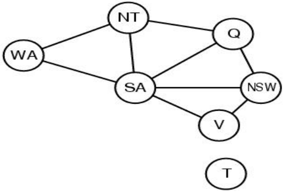
Problem structure



Tasmania and mainland are independent subproblems

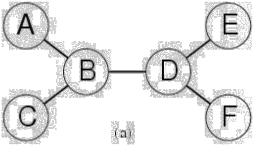
Identifiable as connected components of constraint graph

Graph structure and problem complexity



- Solving disconnected subproblems
  - Suppose each subproblem 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

Graph structure and problem complexity



- Theorem:
  - if a constraint graph has no loops then the CSP can be solved in $O(nd^2)$ time
  - linear in the number of variables!

- Compare difference with general CSP, where worst case is $O(d^n)$