Informatics 2D · Agents and Reasoning · 2019/2020

Lecture 8 · Smart Searching Using Constraints

Claudia Chirita

School of Informatics, University of Edinburgh

30th January 2020

Based on slides by: Jacques Fleuriot, Michael Rovatsos, Michael Herrmann, Vaishak Belle

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSP
- Efficiency matters

Constraint Satisfaction Problems (CSP)

Standard search problem

- A state is a *black box* – any data structure that supports a successor function, a heuristic function and a goal test.

CSP

- A state is defined by a set of variables, each of which has a value.
- Solution: when each variable has a value that satisfies all its constraints.
- Allows useful *general-purpose* algorithms with more power than standard search algorithms.
- Main idea: eliminate large portions of the search space by identifying variable/value combinations that violate the constraints.

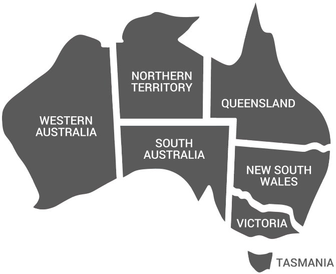
Constraint Satisfaction Problems (CSP)

A CSP consists of:

- a set $X = \{X_1, \dots, X_n\}$ of variables
- a set $D = \{D_1, \dots, D_n\}$ of domains; each domain D_i is a set of possible values for variable X_i
- a set C of constraints that specify accepted combinations of values.

  A constraint $c \in C$ consists of a scope – tuple of variables involved in the constraint – and a relation that defines the values that the variables can take.

Example · Map-Colouring



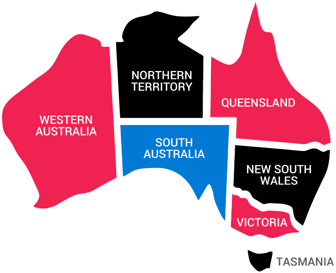
Variables: $\{\mathrm{WA}, \mathrm{NT}, \mathrm{Q}, \mathrm{NSW}, \mathrm{V}, \mathrm{SA}, \mathrm{T}\}$

Domains: $D_i = \{\mathrm{red}, \mathrm{black}, \mathrm{blue}\}$

Constraints: adjacent regions must have different colours

- e.g. $\mathrm{WA} \neq \mathrm{NT}$ or
- $(\mathrm{WA}, \mathrm{NT}) \in \{(\mathrm{red}, \mathrm{black}), (\mathrm{red}, \mathrm{blue}), (\mathrm{black}, \mathrm{red}), (\mathrm{black}, \mathrm{blue}), (\mathrm{blue}, \mathrm{red}), (\mathrm{blue}, \mathrm{black})\}$

Example · Map-Colouring



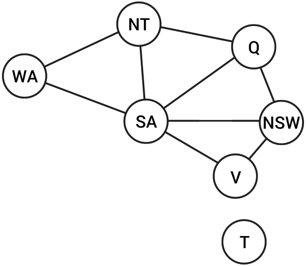
Solutions are complete and consistent assignments.

- e.g. WA $\mapsto$ red, NT $\mapsto$ black, Q $\mapsto$ red, NSW $\mapsto$ black, V $\mapsto$ red, SA $\mapsto$ blue, T $\mapsto$ black.

Constraint graph

Binary CSP

- Each constraint relates two variables.
- Constraint graph:
  - nodes are variables
  - arcs (edges) represent constraints



Varieties of CSP

Discrete variables

- finite domains:
  - n variables, domain size d, $O(d^n)$ complete assignments
  - e.g. Boolean CSPs, including Boolean satisfiability (NP-complete)
- infinite domains:
  - integers, strings, etc.
  - e.g. job scheduling
    - variables are start/end days for each job
    - we need a constraint language to express $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$

Continuous variables

- e.g. start/end times for Hubble Space Telescope observations
- linear constraints solvable in polynomial time by linear programming

Varieties of constraints

Unary constraints involve a single variable.

- e.g. $SA \neq black$

Binary constraints involve pairs of variables.

- e.g. $SA \neq WA$

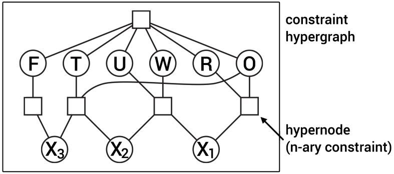
Higher-order constraints involve 3 or more variables.

- e.g. crypt-arithmetic column constraints

Global constraints involve an arbitrary number of variables.

Example · Crypt-arithmetic

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



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: $\text{Alldiff}(F, T, U, W, R, O)$ ⟵ global constraint

$$O + O = R + 10 \cdot X_1$$
$$X_1 + W + W = U + 10 \cdot X_2$$
$$X_2 + T + T = O + 10 \cdot X_3$$
$$X_3 = F, \quad T \neq 0, \quad F \neq 0$$

Real-world CSP

Assignment problems

- e.g. who teaches what class

Timetabling problems

- e.g. which class is offered when and where

Transportation scheduling

Factory scheduling

Many real-world problems involve real-valued variables.

Standard search formulation (incremental)

Let's start with the straightforward approach, then adapt 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 the current assignment
  $\rightarrow$ *fail* if no legal assignments
  Goal test: the current assignment is complete

- This is the same for all CSPs.
- For CSPs with with n variables, any solution appears at depth $n \Rightarrow$ use depth-first search.

Backtracking search

- Variable assignments are commutative.
  - e.g. [$\text{WA} \mapsto \text{red}$ then $\text{NT} \mapsto \text{black}$]

    is the same as

    [$\text{NT} \mapsto \text{black}$ then $\text{WA} \mapsto \text{red}$]
- We only need to consider assignments to a single variable at each node. Thus, $b = d$, and there are d^n leaves.
- Depth-first search for CSPs with single-variable assignments is called backtracking search.
- Backtracking search: the basic uninformed algorithm for CSP.
- Can solve the n-queens problem for $n \approx 25$.

Backtracking search

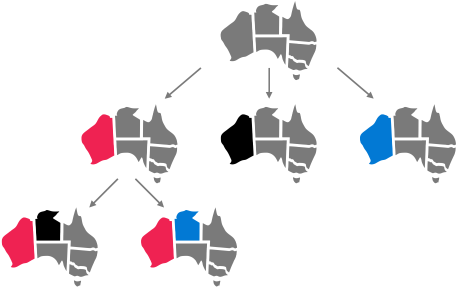
function BACKTRACKING-SEARCH(csp) **returns** a solution, or failure
  return BACKTRACK($\{\ \}$, csp)

function BACKTRACK($assignment$, csp) **returns** a solution, or failure
  if $assignment$ is complete **then return** $assignment$
  $var \leftarrow$ SELECT-UNASSIGNED-VARIABLE(csp)
  for each $value$ **in** ORDER-DOMAIN-VALUES(var, $assignment$, csp) **do**
    if $value$ is consistent with $assignment$ **then**
      add $\{var = value\}$ to $assignment$
      $inferences \leftarrow$ INFERENCE(csp, var, $value$) ← Optional; can be used to impose arc-consistency (more on this later)
      if $inferences \neq failure$ **then**
        add $inferences$ to $assignment$
        $result \leftarrow$ BACKTRACK($assignment$, csp)
        if $result \neq failure$ **then**
          return $result$
    remove $\{var = value\}$ and $inferences$ from $assignment$
  return $failure$

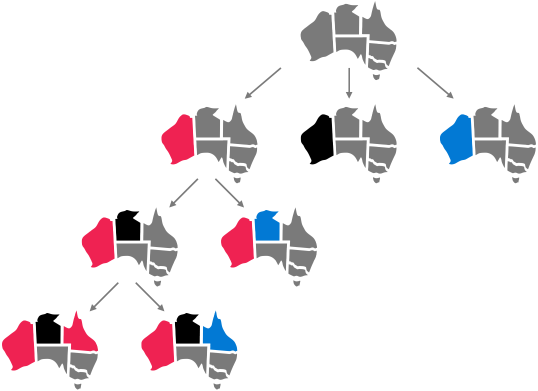
Backtracking example

Backtracking example

Backtracking example



Backtracking example



Improving backtracking efficiency

General-purpose methods can give huge gains in speed.

Which variable should be assigned next?

- SELECT-UNASSIGNED-VARIABLE

Then, in what order should its values be tried?

- ORDER-DOMAIN-VALUES

What inferences should be performed at each search step?

- INFERENCE

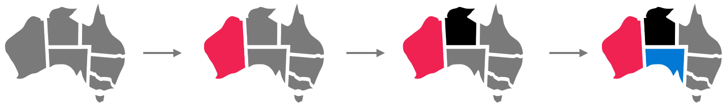
Can we detect inevitable failure early?

Most constrained variable

var ← SELECT-UNASSIGNED-VARIABLE(csp)

Most constrained variable heuristic

- choose the variable with the fewest legal values
- a.k.a. minimum-remaining-values (MRV) heuristic

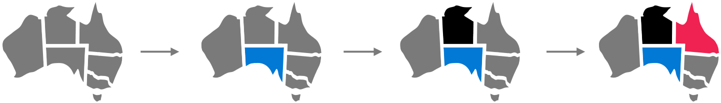


Most constraining variable

Tie-breaker among most constrained variables.

Most constraining variable heuristic

- choose the variable with the most constraints on remaining variables, thus reducing branching
- a.k.a. degree heuristic



Least constraining value

ORDER-DOMAIN-VALUES

Given a variable, choose the least constraining value

- the one that rules out the fewest values in the remaining variables

Combining these heuristics: n-queens feasible for $n \approx 1000$.

Inference · Forward checking

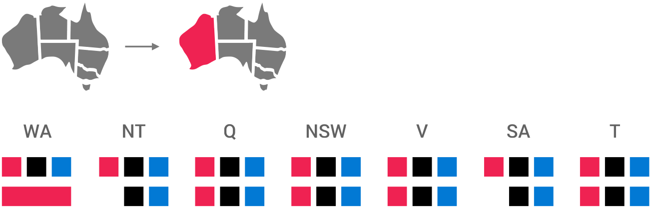
Idea

Keep track of remaining legal values for unassigned variables.
Terminate the search when a variable has no more legal values.

Inference · Forward checking

Idea

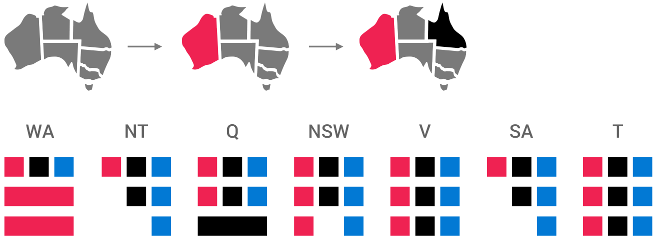
Keep track of remaining legal values for unassigned variables.
Terminate the search when a variable has no more legal values.



Inference · Forward checking

Idea

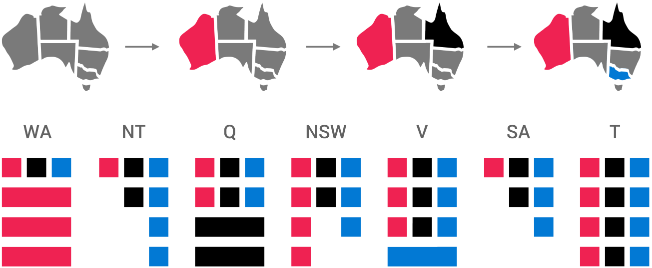
Keep track of remaining legal values for unassigned variables. Terminate the search when a variable has no more legal values.



Inference · Forward checking

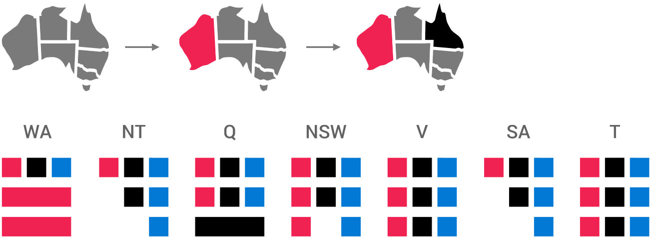
Idea

Keep track of remaining legal values for unassigned variables. Terminate the search when a variable has no more 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

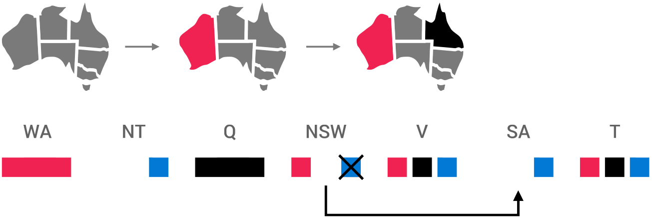
Simplest form of propagation makes each arc consistent.

$X \rightarrow Y$ is consistent iff
  for every value x in the domain of X
  there is some allowed value y in the domain of Y

Arc consistency

Simplest form of propagation makes each arc consistent.

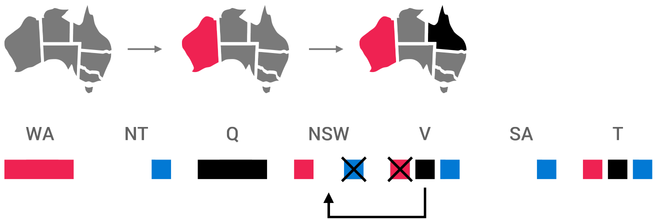
$X \to Y$ is consistent iff
for every value x in the domain of X
there is some allowed value y in the domain of Y



Arc consistency

Simplest form of propagation makes each arc consistent.

$X \to Y$ is consistent iff
for every value x in the domain of X
there is some allowed value y in the domain of Y

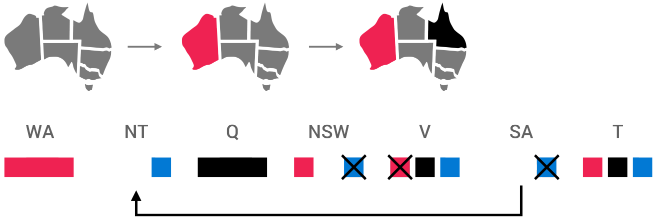


If X loses a value, its neighbours need to be rechecked.

Arc consistency

Simplest form of propagation makes each arc consistent.

$X \rightarrow Y$ is consistent iff
for every value x in the domain of X
there is some allowed value y in the domain of Y



If X loses a value, its neighbours need to be rechecked.
Detects failure earlier than forward checking.
Can be run as a preprocessor or after each assignment.

Arc consistency algorithm · AC-3

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
   inputs: csp, a binary CSP with components (X, D, C)
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (X_i, X_j) ← REMOVE-FIRST(queue)
      if REVISE(csp, X_i, X_j) then                 ← Make X_i arc-consistent with respect to X_j
         if size of D_i = 0 then return false       ← No consistent value left for X_i so fail
         for each X_k in X_i.NEIGHBORS - {X_j} do
            add (X_k, X_i) to queue                 ← Since revision occurred, add all neighbours
                                                      of X_j for consideration (or reconsideration)
   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
```

d – maximum size of the domains

c – number of binary constraints

Time complexity: $O(cd^3)$ Space complexity: $O(c)$

Summary

In CSPs:

- 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.
- Forward checking prevents assignments that are certain to lead to later failure.
- Constraint propagation does additional work to limit values and detect inconsistencies.