CHAPTER 6: CONSTRAINT SATISFACTION PROBLEMS

DIT410/TIN174, Artificial Intelligence

Peter Ljunglöf

31 March, 2017

TABLE OF CONTENTS

CSP: CONSTRAINT SATISFACTION PROBLEMS (R&N 6.1)

FORMULATING A CSP

CONSTRAINT GRAPH

CONSTRAINT SATISFACTION PROBLEMS (CSP)

Standard search problem:

- the *state* is a “black box”,
  any data structure that supports: goal test, cost evaluation, successor

CSP is a more specific search problem:

- the *state* is defined by *variables* X_i, taking values from the domain $\mathbf{D}_i$

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

Since CSP is more specific, it allows useful algorithms with more power than standard search algorithms

STATES AND VARIABLES

Just a few variables can describe many states:

n	binary variables can describe	2^n states
10	binary variables can describe	2^{10} = 1,024
20	binary variables can describe	2^{20} = 1,048,576
30	binary variables can describe	2^{30} = 1,073,741,824
100	binary variables can describe	2^{100} = 1,267,650,600,228,229,401,496,703,205,376

HARD AND SOFT CONSTRAINTS

Given a set of variables, assign a value to each variable that either

- satisfies some set of constraints:
    - *satisfiability problems* — "hard constraints"
- or minimizes some cost function,
  where each assignment of values to variables has some cost:
    - *optimization problems* — "soft constraints" — "preferences"

Many problems are a mix of hard constraints and preferences
(constraint optimization problems)

RELATIONSHIP TO SEARCH

CSP differences to general search problems:

- The path to a goal isn’t important, only the solution is.
- There are no predefined starting nodes.
- Often these problems are huge, with thousands of variables, so systematically searching the space is infeasible.
- For optimization problems, there are no well-defined goal nodes.

FORMULATING A CSP

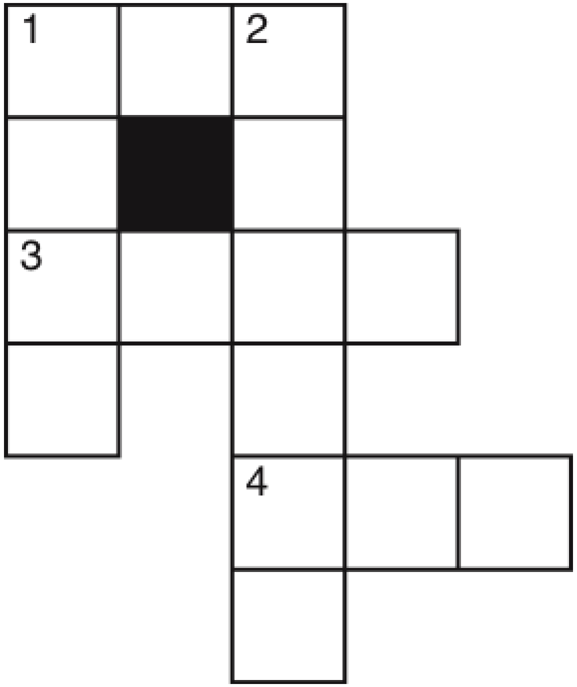
A CSP is characterized by

- A set of variables $X_1, X_2, \ldots, X_n$.
- Each variable X_i has an associated domain $\mathbf{D}_i$ of possible values.
- There are hard constraints $C_{X_i,\ldots,X_j}$ on various subsets of the variables which specify legal combinations of values for these variables.
- A solution to the CSP is an *assignment* of a value to each variable that satisfies all the constraints.

EXAMPLE: SCHEDULING ACTIVITIES

Variables:	A, B, C, D, E representing starting times of various activities. (e.g., courses and their study periods)
Domains:	$\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \mathbf{D}_D = \mathbf{D}_E = \{1, 2, 3, 4\}$
Constraints:	$(B \neq 3), (C \neq 2), (A \neq B), (B \neq C), (C < D), (A = D),$ $(E < A), (E < B), (E < C), (E < D), (B \neq D)$

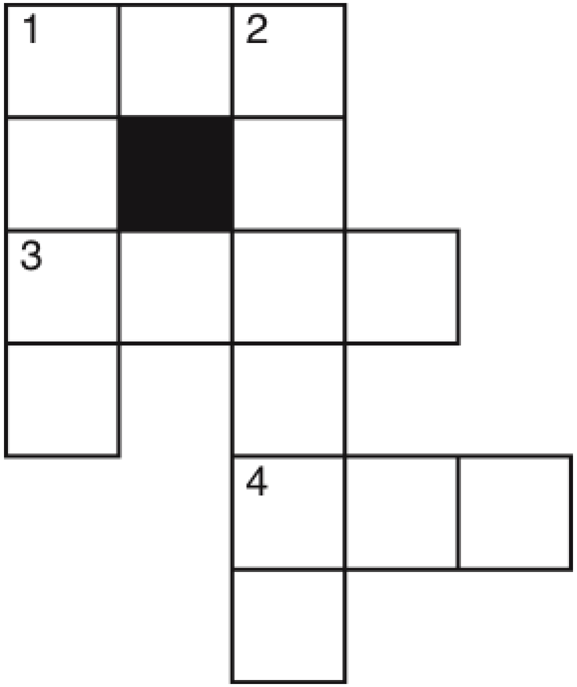
EXAMPLE: CROSSWORD PUZZLE



Words: ant, big, bus, car, has, book, buys, hold, lane, year, beast, ginger, search, symbol, syntax, …

DUAL REPRESENTATIONS

Many problems can be represented in different ways as a CSP, e.g., the crossword puzzle:



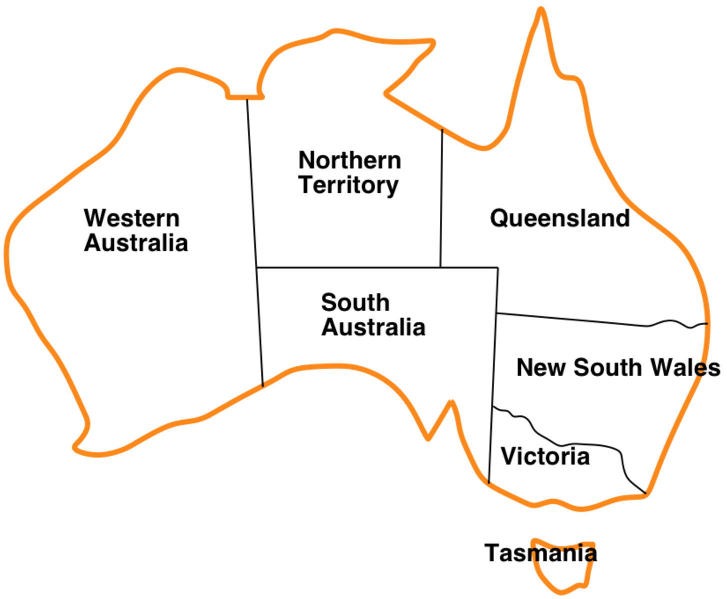
One representation:

- nodes represent word positions: 1-down…6-across
- domains are the words
- constraints specify that the letters on the intersections must be the same

Dual representation:

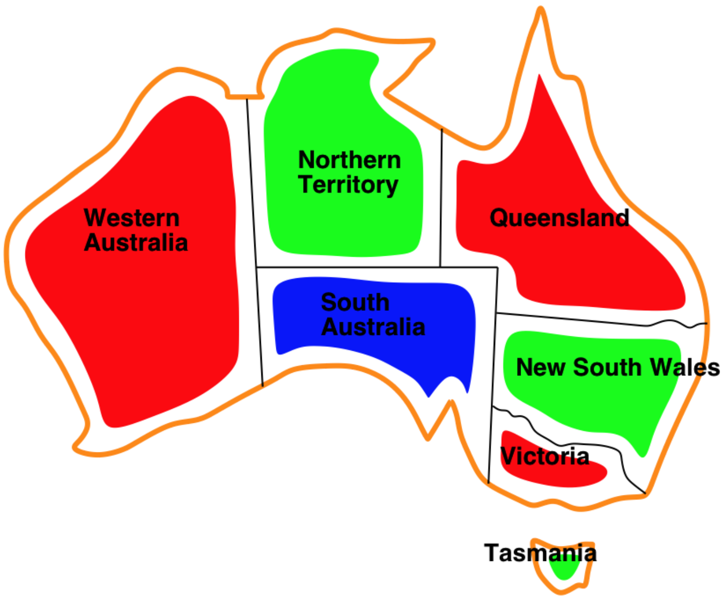
- nodes represent the individual squares
- domains are the letters
- constraints specify that the words must fit

EXAMPLE: MAP COLOURING



Variables:	WA, NT, Q, NSW, V, SA, T
Domains:	$\mathbf{D}_i = \{red, green, blue\}$
Constraints:	adjacent regions must have different colors, i.e., $WA \neq NT, WA \neq SA, NT \neq SA, NT \neq Q, \ldots$

EXAMPLE: MAP COLOURING

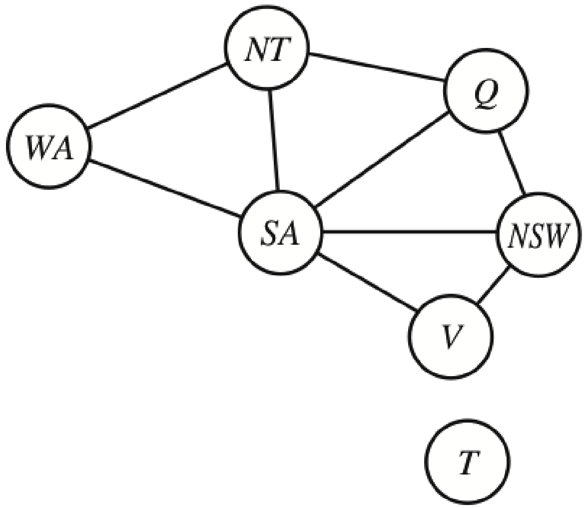


Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green,$
$V = red, SA = blue, T = green\}$

CONSTRAINT GRAPH

Binary CSP: each constraint relates at most two variables
(note: this does not say anything about the domains)

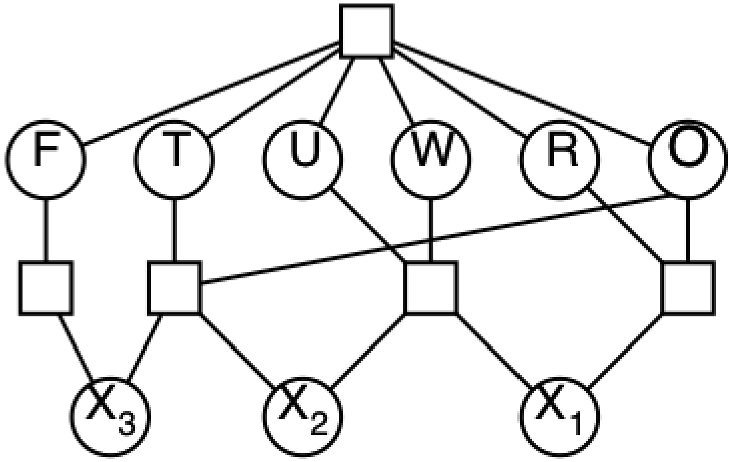
Constraint graph: every variable is a node, every binary constraint is an arc



CSP algorithms can use the graph structure to speed up search,
e.g., Tasmania is an independent subproblem.

EXAMPLE: CRYPTARITHMETIC PUZZLE

```
    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:	$\mathit{Alldiff}(F, T, U, W, R, O)$, $O + O = R + 10 \cdot X_1$, etc.
Note:	This is not a binary CSP! The graph is a *constraint hypergraph*

EXAMPLE: SUDOKU

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

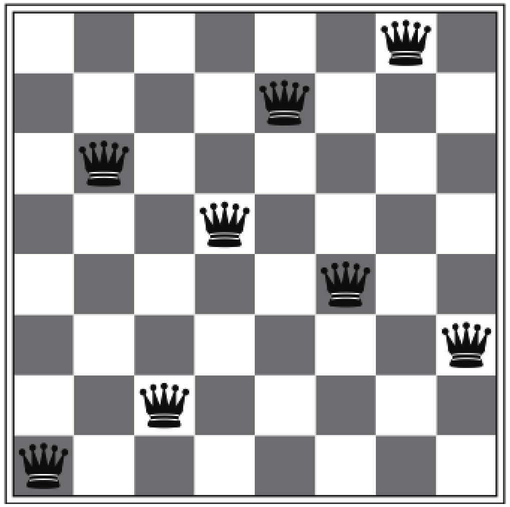
(a)

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

(b)

Variables:	$A_1 \ldots A_9, B_1, \ldots, E_5, \ldots, I_9$
Domains:	$\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraints:	$Alldiff(A_1, \ldots, A_9), \ldots, Alldiff(A_5, \ldots, I_5), \ldots, Alldiff(D_1, \ldots, F_3), \ldots,$ $B_1 = 9, \ldots, F_6 = 8, \ldots, I_7 = 3$

EXAMPLE: N-QUEENS



Variables:	$Q_1, Q_2, \ldots, Q_n$
Domains:	$\{1, 2, 3, \ldots, n\}$
Constraints:	$\mathit{Alldiff}(Q_1, Q_2, \ldots, Q_n)$, $Q_i - Q_j \neq \lvert i - j \rvert \quad (1 \leq i < j \leq n)$

CSP VARIETIES

Discrete variables, *finite domains*:

- n variables, domain size $d \Rightarrow O(d^n)$ complete assignments
- what we discuss in this course

Discrete variables, *infinite domains* (integers, strings, etc.)

- e.g., job scheduling — variables are start/end times for each job
- we need a *constraint language* for formulating the constraints (e.g., $T_1 + d_1 \leq T_2$)
- *linear* constraints are solvable — *nonlinear* are undecidable

Continuous variables:

- e.g., scheduling for Hubble Telescope observations and manouvers
- linear constraints (*linear programming*) — solvable in polynomial time!

DIFFERENT KINDS OF CONSTRAINTS

Unary constraints involve a single variable:

- e.g., $SA \neq green$

Binary constraints involve pairs of variables:

- e.g., $SA \neq WA$

Global constraints (or *higher-order*) involve 3 or more variables:

- e.g., $Alldiff(WA, NT, SA)$
- all global constraints can be reduced to a number of binary constraints (but this might lead to an explosion of the number of constraints)

Preferences (or soft constraints):

- "constraint optimization problems"
- often representable by a cost for each variable assignment
- not discussed in this course

CSP AS A SEARCH PROBLEM (R&N 6.3–6.3.2)

BACKTRACKING SEARCH

HEURISTICS: IMPROVING BACKTRACKING EFFICIENCY

GENERATE-AND-TEST ALGORITHM

Generate the assignment space $\mathbf{D} = \mathbf{D}_{V_1} \times \mathbf{D}_{V_2} \times \cdots \times \mathbf{D}_{V_n}$
Test each assignment with the constraints.

Example:

$$
\begin{aligned}
\mathbf{D} &= \mathbf{D}_A \times \mathbf{D}_B \times \mathbf{D}_C \times \mathbf{D}_D \times \mathbf{D}_E \\
&= \{1, 2, 3, 4\} \times \cdots \times \{1, 2, 3, 4\} \\
&= \{(1, 1, 1, 1, 1), (1, 1, 1, 1, 2), \ldots, (4, 4, 4, 4, 4)\}
\end{aligned}
$$

How many assignments need to be tested for n variables, each with domain size $d = |\mathbf{D_i}|$?

CSP AS A SEARCH PROBLEM

Let's start with the straightforward, dumb approach.

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 there are no legal assignments
- *Goal test*: the current assignment is complete

Every solution appears at depth n (assuming n variables)
$\Longrightarrow$ we can use depth-first-search, no risk for infinite loops

At search depth k, the branching factor is $b = (n-k)d$
(where $d = |\mathbf{D}_i|$ is the domain size and $n-k$ is the number of unassigned variables)
$\Longrightarrow$ hence there are $n!d^n$ leaves

BACKTRACKING SEARCH

Variable assignments are commutative:

- $\{WA = red, NT = green\}$ is the same as $\{NT = green, WA = red\}$

It's unnecessary work to assign WA followed by NT in one branch, and NT followed by WA in another branch.

Instead, at each depth level, we can decide on one single variable to assign:

- this gives branching factor $b = d$, so there are d^n leaves (instead of $n!d^n$)

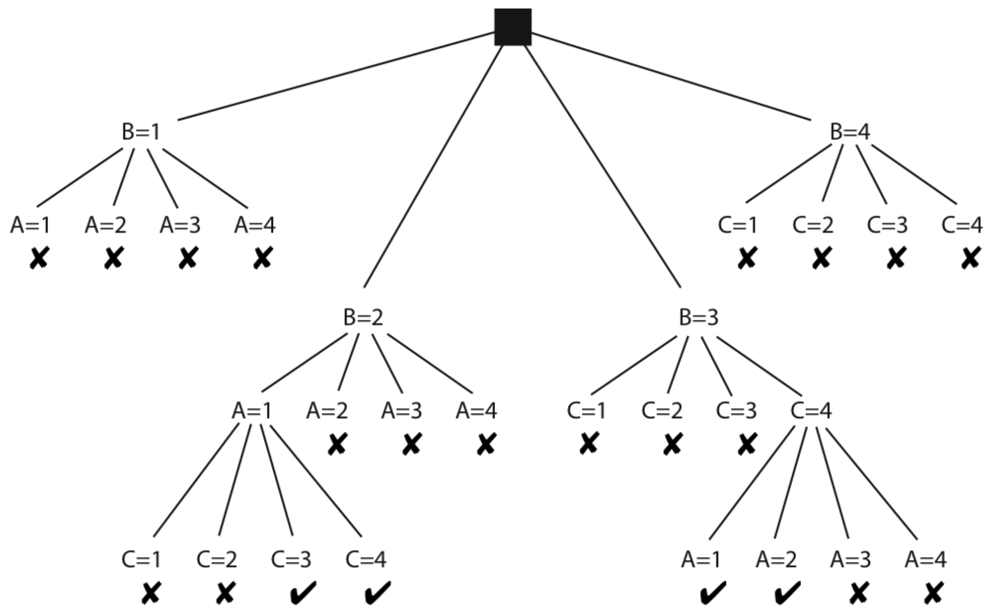
Depth-first search with single-variable assignments is called *backtracking search*:

- backtracking search is the basic uninformed CSP algorithm
- it can solve n-queens for $n \approx 25$

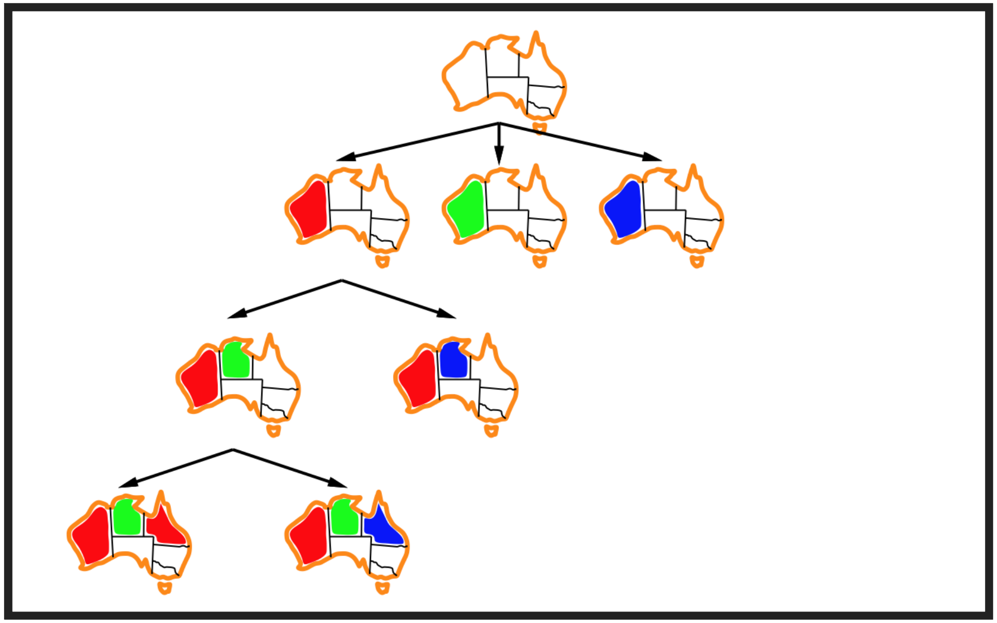
Why not use breadth-first search?

SIMPLE BACKTRACKING EXAMPLE

Variables:	A, B, C
Domains:	$\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \{1, 2, 3, 4\}$
Constraints:	$(A < B), (B < C)$



EXAMPLE: AUSTRALIA MAP COLOURS



Assign variable: Q

ALGORITHM FOR BACKTRACKING SEARCH

```
function BacktrackingSearch(csp):
    return Backtrack(csp, { })

function Backtrack(csp, assignment):
    if assignment is complete then return assignment
    var := SelectUnassignedVariable(csp, assignment)
    for each value in OrderDomainValues(csp, var, assignment):
        if value is consistent with assignment:
            inferences := Inference(csp, var, value)
            if inferences ≠ failure:
                result := Backtrack(csp, assignment ∪ {var=value} ∪ inferences)
                if result ≠ failure then return result
    return failure
```

HEURISTICS: IMPROVING BACKTRACKING EFFICIENCY

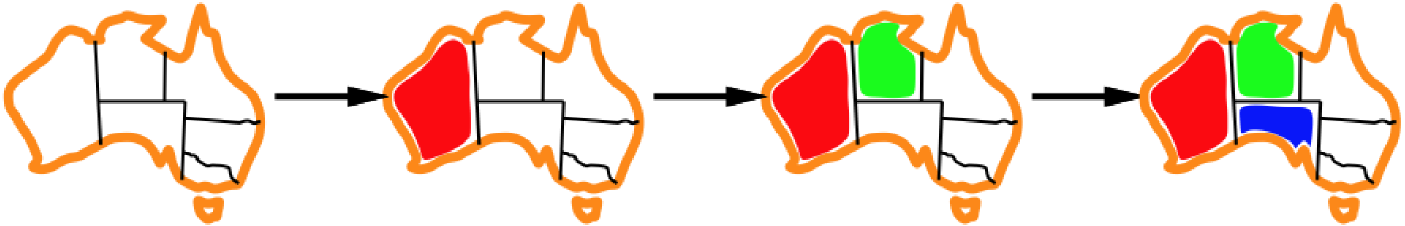
The general-purpose algorithm gives rise to several questions:

- Which variable should be assigned next?
  - $\text{SelectUnassignedVariable}(csp, assignment)$
- In what order should its values be tried?
  - $\text{OrderDomainValues}(csp, var, assignment)$
- What inferences should be performed at each step?
  - $\text{Inference}(csp, var, value)$
- Can the search avoid repeating failures?
  - Conflict-directed backjumping, constraint learning, no-good sets (R&N 6.3.3, not covered in this course)

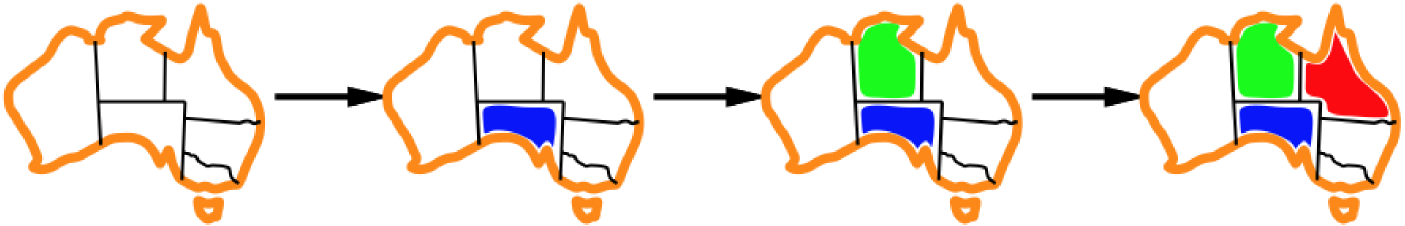
SELECTING UNASSIGNED VARIABLES

Heuristics for selecting the next unassigned variable:

- Minimum remaining values (MRV):
  $\Longrightarrow$ choose the variable with the fewest legal values



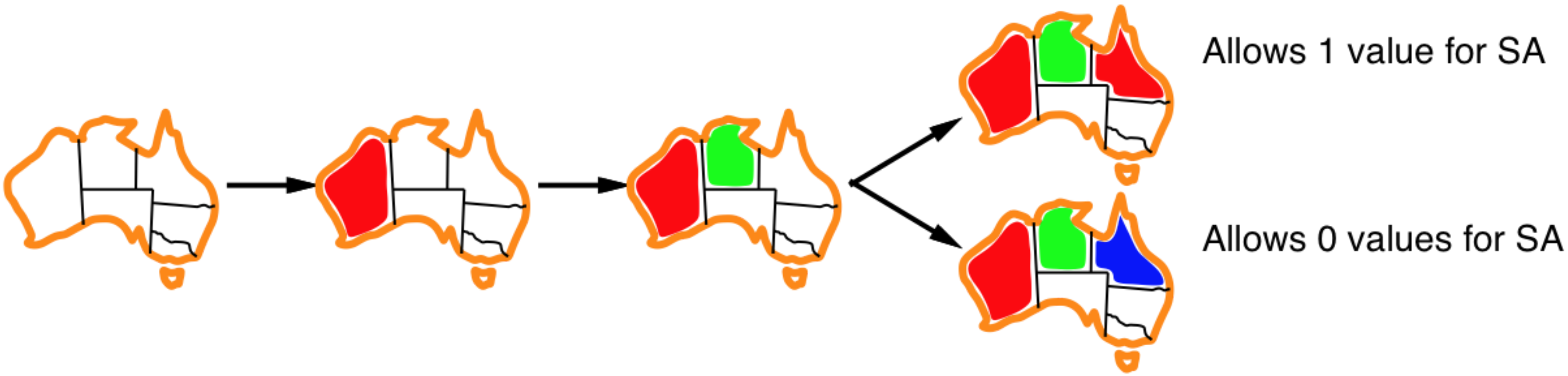
- Degree heuristic (if there are several MRV variables):
  $\Longrightarrow$ choose the variable with most constraints on remaining variables



ORDERING DOMAIN VALUES

Heuristics for ordering the values of a selected variable:

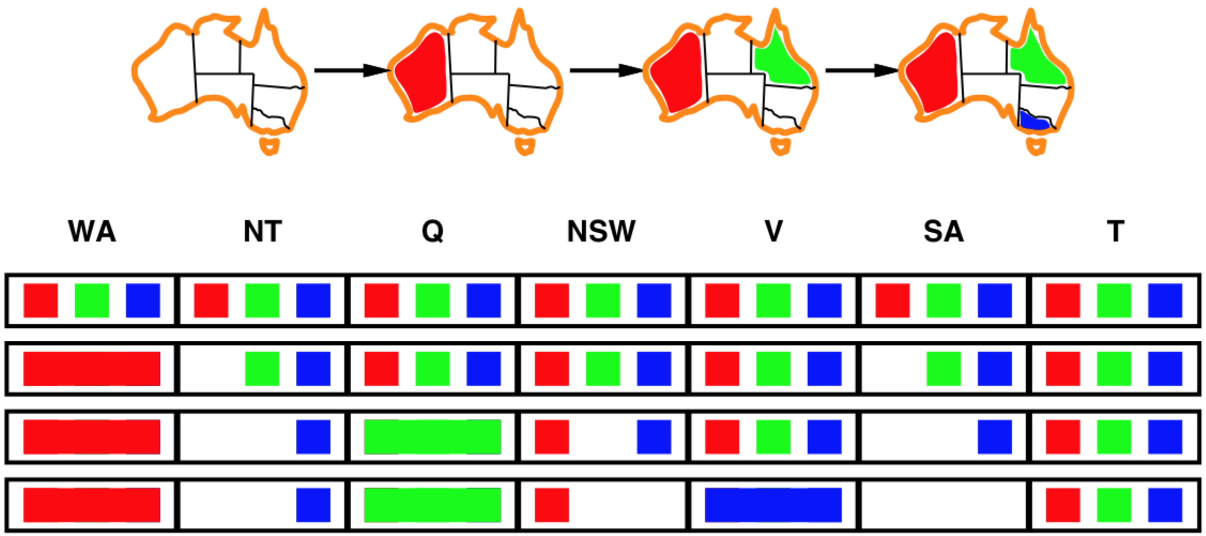
- Least constraining value:
  $\Longrightarrow$ prefer the value that rules out the fewest choices for the neighboring variables in the constraint graph



INFERENCE: FORWARD CHECKING

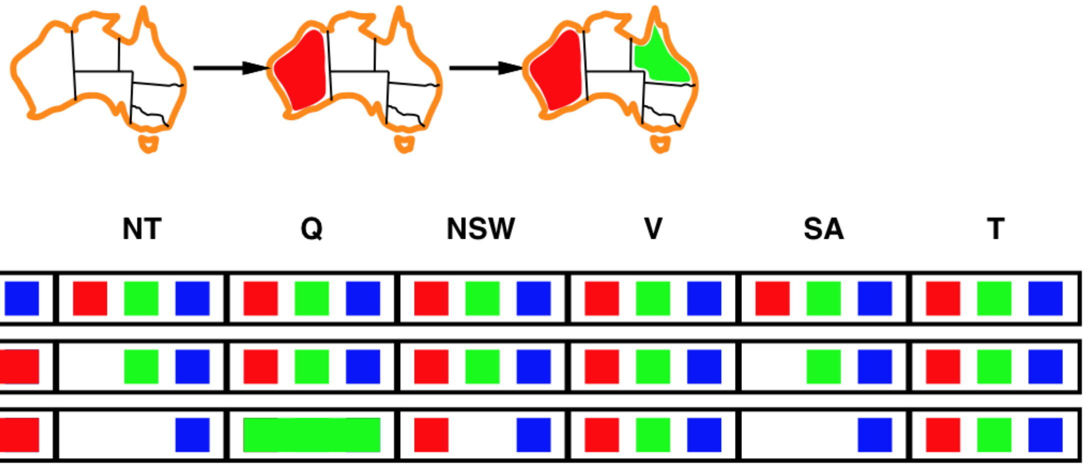
Forward checking is a simple form of inference:

- Keep track of remaining legal values for unassigned variables
  — terminate when any variable has no legal values left
- When a new variable is assigned, recalculate the legal values for its neighbors



INFERENCE: CONSTRAINT PROPAGATION

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



NT and SA cannot both be blue!

- *Forward checking* enforces local constraints
- *Constraint propagation* enforces local constraints, *repeatedly* until reaching a fixed point

CONSTRAINT PROGAGATION (R&N 6.2–6.2.2)

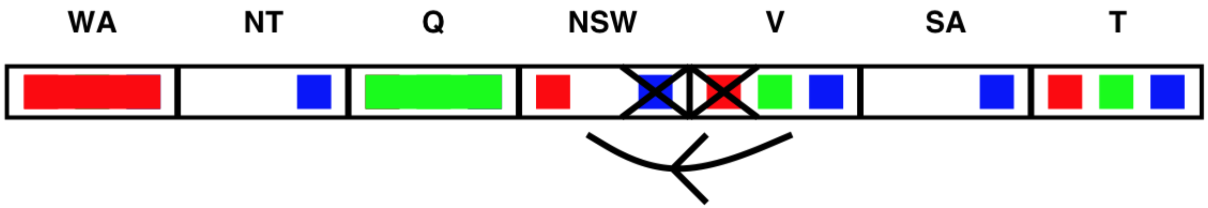
ARC CONSISTENCY

MAINTAINING ARC CONSISTENCY

CONSTRAINT PROPAGATION: ARC CONSISTENCY

The simplest form of propagation is to make each arc consistent:

- $X \rightarrow Y$ is *arc consistent* iff:
  for every value x of X, there is some allowed value y in Y



- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking

CONSISTENCY

Different variants of constistency:

- A variable is *node-consistent* if all values in its domain satisfy its own unary constraints,
- a variable is *arc-consistent* if every value in its domain satisfies the variable's binary constraints,
- *Generalised arc-consistency* is the same, but for n-ary constraints,
- *Path consistency* is arc-consistency, but for 3 variables at the same time.
- k*-consistency* is arc-consistency, but for k variables,
- …and there are consistency checks for several global constraints, such as $Alldiff$ and $Atmost$.

A network is X-consistent if every variable is X-consistent with every other variable.

SCHEDULING EXAMPLE (AGAIN)

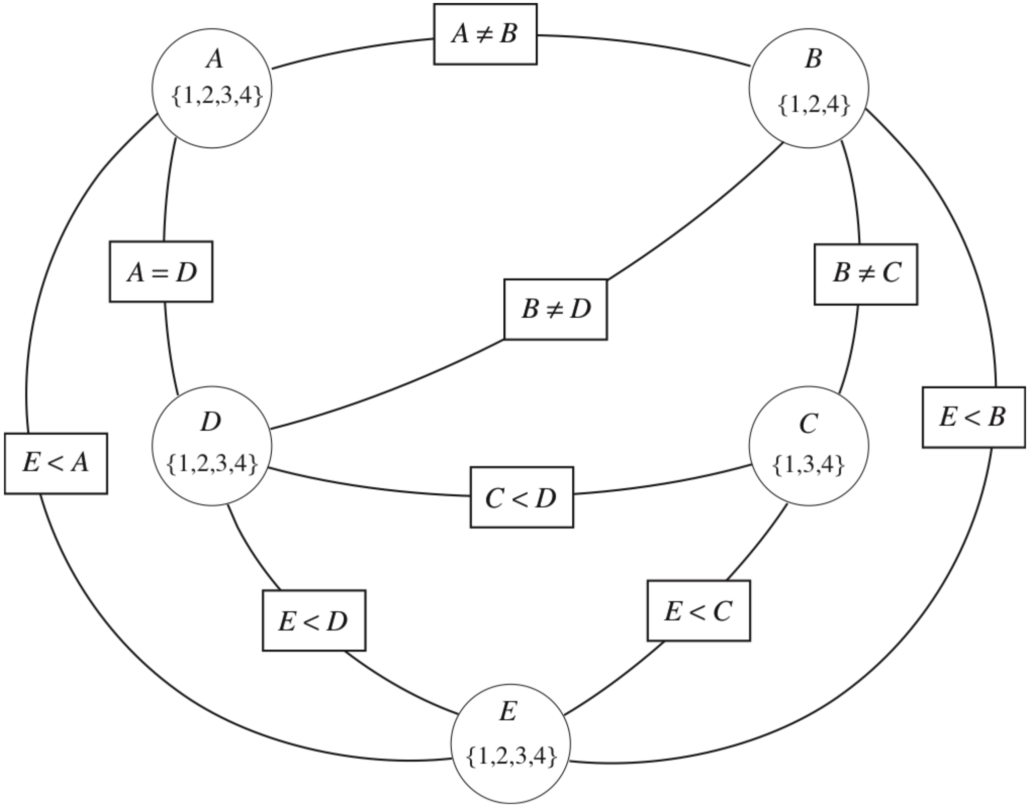
Variables:	A, B, C, D, E representing starting times of various activities.
Domains:	$\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \mathbf{D}_D = \mathbf{D}_E = \{1, 2, 3, 4\}$
Constraints:	$(B \neq 3), (C \neq 2), (A \neq B), (B \neq C), (C < D), (A = D),$ $(E < A), (E < B), (E < C), (E < D), (B \neq D)$

Is this example node consistent?

- $\mathbf{D}_B = \{1, 2, 3, 4\}$ is not node consistent, since $B = 3$ violates the constraint $B \neq 3$ $\Longrightarrow$ reduce the domain $\mathbf{D}_B = \{1, 2, 4\}$
- $\mathbf{D}_C = \{1, 2, 3, 4\}$ is not node consistent, since $C = 2$ violates the constraint $C \neq 2$ $\Longrightarrow$ reduce the domain $\mathbf{D}_C = \{1, 3, 4\}$

SCHEDULING EXAMPLE AS A CONSTRAINT GRAPH

If we reduce the domains for B and C, then the constraint graph is node consistent.



ARC CONSISTENCY

A variable X is binary *arc-consistent* with respect to another variables (Y) if:

- For each value $x \in \mathbf{D}_X$, there is some $y \in \mathbf{D}_Y$ such that the binary constraint $C_{XY}(x, y)$ is satisfied.

A variable X is *generalised arc-consistent* with respect to variables $(Y, Z, \ldots)$ if:

- For each value $x \in \mathbf{D}_X$, there is some assignment $y, z, \cdots \in \mathbf{D}_Y, \mathbf{D}_Z, \ldots$ such that $C_{XYZ\ldots}(x, y, z, \ldots)$ is satisfied.

What if X is not arc consistent to Y?

- All values $x \in \mathbf{D}_X$ for which there is no corresponding $y \in \mathbf{D}_Y$ can be deleted from $\mathbf{D}_X$ to make X arc consistent.

Note! The arcs in a constraint graph are directed:

- (X, Y) and (Y, X) are considered as two different arcs,
- i.e., X can be arc consistent to Y, but Y not arc consistent to X.

ARC CONSISTENCY ALGORITHM

Keep a set of arcs to be considered: pick one arc (X, Y) at the time and make it consistent (i.e., make X arc consistent to Y).

- Start with the set of all arcs $\{(X, Y), (Y, X), (X, Z), (Z, X), \dots\}$.

When an arc has been made arc consistent, does it ever need to be checked again?

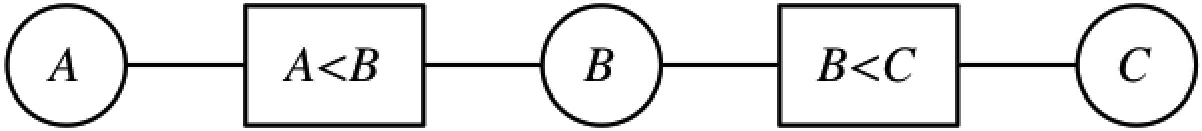
- An arc (X, Y) needs to be revisited if the domain of Y is revised.

Three possible outcomes when all arcs are made arc consistent:
(Is there a solution?)

- One domain is empty $\Longrightarrow$ *no solution*
- Each domain has a single value $\Longrightarrow$ *unique solution*
- Some domains have more than one value $\Longrightarrow$ *maybe a solution, maybe not*

QUIZ: ARC CONSISTENCY

The variables and constraints are in the constraint graph:



Assume the initial domains are $\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \{1, 2, 3, 4\}$

How will the domains look like after making the graph arc consistent?

THE ARC CONSISTENCY ALGORITHM AC-3

```
function AC-3(inout csp):
    initialise queue to all arcs in csp
    while queue is not empty:
        (X, Y) := RemoveOne(queue)
        if Revise(csp, X, Y):
            if  D_X = ∅  then return false
            for each Z in X.neighbors–{Y}:
                add (Z, X) to queue
    return true

function Revise(inout csp, X, Y):
    revised := false
    for each x in D_X:
        if there is no value y in D_Y satisfying the csp constraint C_XY(x, y):
            delete x from D_X
            revised := true
    return revised
```

Note: This algorithm destructively updates the domains of the CSP! You might need to copy the CSP before calling AC-3.

MAINTAINING ARC-CONSISTENCY (MAC)

What if some domains have more than one element after AC?

We can always resort to backtracking search:

- Select a variable and a value using some heuristics
  (e.g., minimum-remaining-values, degree-heuristic, least-constraining-value)
- Make the graph arc-consistent again
- Backtrack and try new values/variables, if AC fails
- Select a new variable/value, perform arc-consistency, etc.

Do we need to restart AC from scratch?

- no, only some arcs risk becoming inconsistent after a new assignment
- restart AC with the queue $\{(Y_i, X) | X \rightarrow Y_i\}$,
  i.e., only the arcs (Y_i, X) where Y_i are the neighbors of X
- this algorithm is called *Maintaining Arc Consistency* (MAC)

DOMAIN SPLITTING (NOT IN R&N)

What if some domains are very big?

- Instead of assigning every possible value to a variable, we can split its domain
- Split one of the domains, then recursively solve each half, i.e.:
    - perform AC on the resulting graph, then split a domain, perform AC, split a domain, perform AC, split, etc.
- It is often good to split a domain in half, i.e.:
    - if $\mathbf{D}_X = \{1, \ldots, 1000\}$, split into $\{1, \ldots 500\}$ and $\{501, \ldots, 1000\}$