Constraint satisfaction problems (CSP)

Solutions with caveats

a warning of conditions

Example: Find a way to take classes such that I graduate in four years

- prerequisites
- course availability
- funding

Constraint satisfaction problems (CSP)

- To date, states were
  - atomic – didn't care about internal representation except with respect to analyzing for goal/heuristic
  - mutated by actions that produced a new atomic state
- Factored representations

all important

  - states have internal structure
  - structure can be manipulated
  - constraints relate different parts of the structure to one another and provide legal/illegal configurations

3

CSP Definition

Problem = {X, D, C}

- X – Set of variables $X = \{X_1, X_2, \ldots, X_n\}$
- D – Set of domains $D = \{D_1, D_2, \ldots, D_n\}$
  such that $X_i = x_i$ where $x_i \in D_i$
- C – Set of constraints $C = \{C_1, C_2, \ldots, C_m\}$
  such that $C_i = \langle (C_a, C_b), \text{relationship}(C_a, C_b) \rangle$

rule

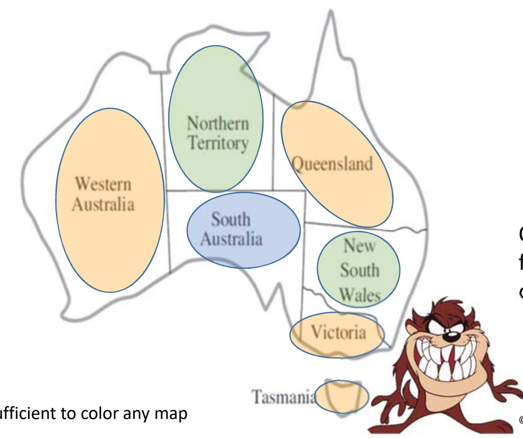
$C_a > C_b$

Set of constraints that specify allowable combinational values

4

CSP example: map coloring

- Color territories on a map using 3 colors such that no two colors are adjacent

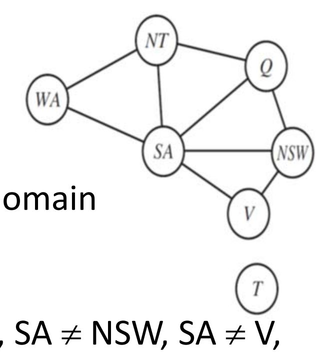


One possible solution for colors:
orange, blue, and green

Note: 4 colors are sufficient to color any map

© Warner Bros

Map coloring



- Graph representation
- Variables
  X={WA, NT, SA, Q, NSW, V, T}
- All variables have the same domain
  D_i={red, green, blue}
- Constraint set
  C={SA $\neq$ WA, SA $\neq$ NT, SA $\neq$ Q, SA $\neq$ NSW, SA $\neq$ V,
  WA $\neq$ NT, NT $\neq$ Q, Q $\neq$ NSW, NSW $\neq$ V}
  or $\{\underbrace{\text{adjacent}(t_a,t_b)}_{\text{Function}} \rightarrow t_a \neq t_b\}$

Scheduling example

Partial auto assembly

- Install front and rear axels (10 m each)
- Install four wheels (1 m each)
- Install nuts on wheels (2 m each wheel)
- Attach hubcap (1 m each)
- Inspect

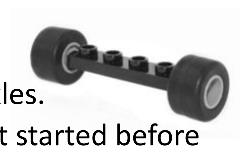
Sequentially

Variable set X

$$X = \begin{Bmatrix} Axle_F, & Axle_B, & Wheel_{RF}, & Wheel_{LF}, \\ Wheel_{RB}, & Wheel_{LB}, & Nuts_{RF}, & Nuts_{LF}, \\ Nuts_{RB}, & Nuts_{LB}, & Cap_{RF}, & Cap_{LF}, \\ Cap_{RB}, & Cap_{LB}, & Inspect & \end{Bmatrix}$$

Constraint types

- Domain values
  - Time at which task begins {0, 1, 2, …}
- Precedence constraints
  - Suppose it takes 10 minutes to install axles.
  - We can ensure that front wheels are not started before axel assembly is completed:



$$Axle_F + 10 \le Wheel_{RF}$$
$$Axle_F + 10 \le Wheel_{LF}$$

- Disjunctive constraints – e.g. doohickey needed to assemble axle, but only have one

$$Axle_F + 10 \le Axle_B \text{ or } Axle_B + 10 \le Axle_F$$

Constraint types

- Unary – single variable $Z \leq 10$
- Binary – between two variables $Z^2 > Y$
- Global – constraints with 3+ variables can be reduced to multiple binary/unary constraints

$$X \leq Y \leq Z \rightarrow X \leq Y \text{ and } Y \leq Z$$

$$alldiff\,(W, X, Y, Z) \rightarrow W \neq X, W \neq Y, W \neq Z, X \neq Y, \ldots$$

Note: Global constraints do not have to involve *all* variables

9

Constraint graphs

Cryptoarithmetic puzzle
Find digit for each letter
such that problem is valid

$$\begin{array}{cccc} C_3 & C_2 & C_1 & \\ & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

CSP specification O_{C0}

- X = {F,T,U,W,R,O,C_1,C_2,C_3}
- D = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- C = {
  - $O + O = R + 10\,C_1$
  - $C_1 + W + W = U + 10\,C_2$
  - $C_2 + T + T = O + 10C_3$
  - $C_3 = F$
  - Alldiff(F,T,U,W,R,O)

  }

C_i's are auxiliary variables for carry digits

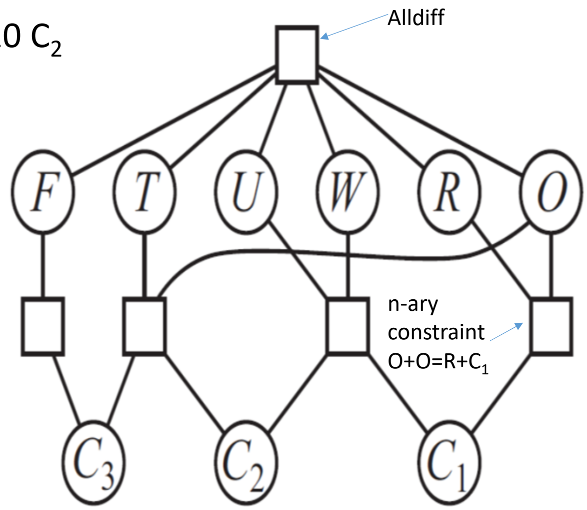
Alldiff(X_1, X_2,…, X_i) → $\forall j, k: j \neq k \text{ and } 1 \leq j, k \leq i, x_j \neq x_k$

10

Constraint hypergraphs

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

$O + O = R + 10\ C_1$
$C_1 + W + W = U + 10\ C_2$
$C_2 + T + T = O + C_3$
$C_3 = F$
Alldiff(F,T,U,W,R,O)



Binarization of constraints

- Convert n-ary constraints into unary/binary ones.
- Example: constraint on X, Y, Z with domains:

$$X \in \{1,2\} Y \in \{3,4\}, Z \in \{5,6\}$$

- Create *encapsulated variable U*
  Cartesian product $U = X \times Y \times Z$

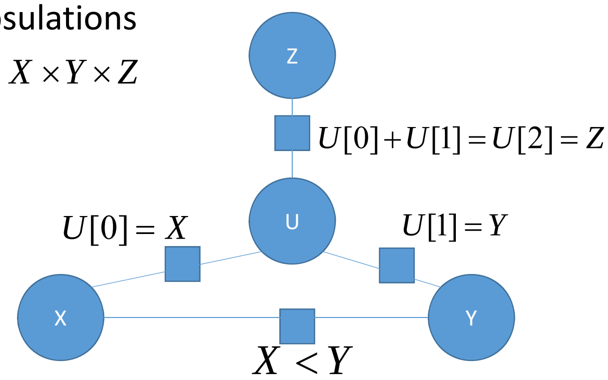
$$U \in \left\{ \begin{array}{l} (1,3,5),(1,3,6),(1,4,5),(1,4,6), \\ (2,3,5),(2,3,6),(2,4,5),(2,4,6) \end{array} \right\}$$

Equivalent binary CSP

- Constraints: $X + Y = Z$

  $X < Y$
- Encapsulations

  $U \triangleq X \times Y \times Z$



Another example: House puzzle

- A row of 5 houses, each one
  - has a color
  - contains a person with a nationality
  - has a household favorite candy
  - has a household favorite drink
  - contains a pet

  - all attributes are distinct
- How should we represent this?

House Puzzle Constraints

- The Englishman lives in the red house.
- The Spaniard owns the dog.
- The Norwegian lives in the first house on the left.
- The green house is immediately to the right of the ivory house.
- The man who eats Hershey bars lives in the house next to the man with the fox.
- Kit Kats are eaten in the yellow house.
- The Norwegian lives next to the blue house.
- The Smarties eater owns snails.

15

House Puzzle Constraints

- The Snickers eater drinks orange juice.
- The Ukranian drinks tea.
- The Japanese eats Milky Ways
- Kit Kats are eaten in a house next to where the horse is kept.
- Coffee is drunk in the green house.
- Milk is drunk in the middle house.

Answer the questions:

Where does the zebra live?

Which house drinks water?

16

House Puzzle Representation

- Variables – What’s common to each thing?
- Domains – What are the domains?

17

House Puzzle representation

- Constraints are location based, e.g. milk is drunk in the middle house.
- Could we associate variables with a location?
- If so, what are
  - our variables?
  - their domains?
  - and how do we write our constraints?

18

House puzzle representation

- Colors: red, green, ivory, yellow, & blue
- Nationalities: English, Spaniard, Norwegian, Ukranian, and Japanese
- Pets: dog, fox, snails, horse, and *zebra*
- Candies: Hershey bars, Kit Kats, Smarties, Snickers, and Milky Way
- Drinks: orange juice, tea, coffee, milk, and *water*

Note: water and zebra were inferred from the questions

19

House puzzle representation

Some examples

- Milk is drunk in the middle house.
  milk = 3
- Coffee is drunk in the green house
  coffee = green
- Kit Kats are eaten in a house next to where the horse is kept.
  abs(kit kats – horse) = 1
- The green house is immediately to the right of the ivory home.
  green = ivory + 1
- The Norwegian lives next to the blue house
  Norwegian = blue + 1 or Norwegian = blue – 1
- The Norwegian lives in the first house on the left
  Norwegian = 1 → blue = 2

20

Implementing a CSP problem: Representation

- variables – simple list
- values – Mapping from variables to value lists
  e.g. Python dictionary w
- neighbors – Mapping from variables to list of other variables that participate in constraints
- binary constraints
  - explicit value pairs
  - functions that return a boolean value

21

Representation of house problem

- variables:
  list of colors, nationalities, pets, candies, & drinks
  {red, green, ivory, yellow, blue, English, Spaniard, …}
- values: $X_i \in \{1,2,4,5\}$
  except milk = {3}, Norwegian = {1}
- neighbors:
  - all variable pairs from constraints, e.g. Englishman & red
  - alldiff(red, green, ivory, blue), alldiff(English, Spaniard, …), other category alldiffs

22

Representation of house problem

- constraints – Function f(A, a, B, b)

  where A and B are variables with values a and b respectively.

  Returns true if constraint is satisfied, otherwise false

  Example: f("Englishman", 4, "red", 5) returns false as the Englishman lives in the red house.

23

How do we tame this beastie?

General strategies

- *Local consistency*: Reduce the set of possible values through constraint enforcement and propagation
  - node consistency
  - arc consistency
  - path consistency
- Perform search on remaining possible states

24

Node consistency

- A variable is *node-consistent* if all values satisfy all unary constraints

$$fruits = \begin{Bmatrix} apples, oranges, strawberries, \\ peaches, pineapple, bananas \end{Bmatrix}$$

$$\text{Condition: } allergic(TreeBornFruit)$$

$$\text{Reduced domain: } \{strawberries, pineapple\}$$

- Other unary conditions could further restrict the domain

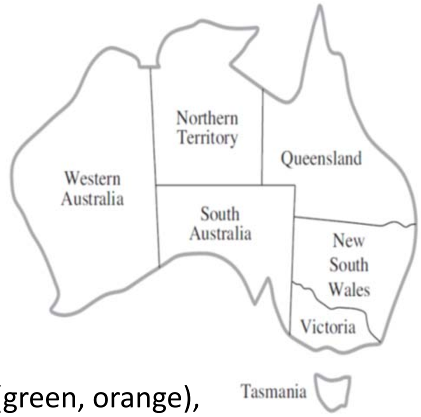
25

Arc consistency

- *arc-consistent*
  - variable - all binary constraints are satisfied for the variable
  - network – all variables in CSP are arc-consistent
- Arc consistency only helps when some combinations of values preclude others…

26

Arc Consistency



Each territory has domain
{orange, green, blue}

WA ≠ SA:
{(orange, green), (orange, blue), (green, orange), (green, blue), (blue, orange), (blue, green)}

Does this reduce the domain of WA or SA?

27

Arc Consistency

- Constraints that eliminate part of the domain can improve arc consistency
- Variables that represent task starting times

T1 = {0, 1, 2, 3, 4, 5, 6, 7 , 8, 9}

T2 = {2, 3, 4, 5, 6, 7, 8, 9}

?

- Constraint: T1 + 5 < T2 yield consistent domains

T1 = {0, 1, 2, 3, 4}

T2 = {6, 7, 8, 9}

28

AC-3 arc consistency algorithm

```
AC3(CSP):  reasoning
  "CSP(variables X, domains D, constraints C)"
  q = Queue(binary arcs in CSP)
  while not q.empty():
           var neighbors
    (X_i, X_j) = q.dequeue()  # get binary constraint
var  if revise(CSP, X_i, X_j):
0      if D_i = ∅ return False
       else:
        for each X_k in neighbors(X_i)- X_j:
          q.enqueue(X_k, X_i)
  return True
```

$O(cd^3)$ worst case complexity (c # constraints, d max domain size)

29

AC-3 arc consistency

```
revise(CSP, X_i, X_j)
  revised = False
  for each x in D_i:
    if not ∃y∈D_j such that constraint holds between x & y:
     delete x from Di
     revised = True

  return revised
```

30

Path and k- consistency

- Higher levels of consistency, beyond our scope
- General ideas:
  - Path consistency
    See if a pair of variables $\{X_i, X_j\}$ consistent with a 3rd variable X_k. Solved similarly to arc consistency
  - K-consistency
    Given k-1 consistent variables, can we make a k^{th} variable consistent (generalization of consistency)

31

Global constraints

Consider the “all different” constraint.

- Each variable has to have a distinct value.
- Assume m variables, and n distinct values.
- What happens when m > n?

32

Global constraints

Extending this idea:

- Find variables constrained to a single value
- Remove these variables and their values from all variables.
- Repeat until no variable is constrained to a single value
- Constraints cannot be satisfied if
    1. A variable remains with an empty domain
    2. There are more variables than remaining values

33

Resource constraints ("atmost")

$$atmost(20, X, Y, Z) \rightarrow X + Y + Z \leq 20$$

$$atmost(10, P_1, P_2, P_3, P_4) \rightarrow \sum_{i=1}^{4} P_i \leq 10$$

- Consistency checks
    - Minimum values of domains satisfy constraints?
    - P_i = {3, 4, 5, 6}
- Domain restriction
    - Are the largest values consistent with the minimum ones?
    - P_i = {2, 3, 4, ~~5, 6~~}

34

Range bounds

- Impractical to store large integer sets
- Ranges can be used [min, max] instead
- Bounds propagation can be used to restrict domains according to constraints

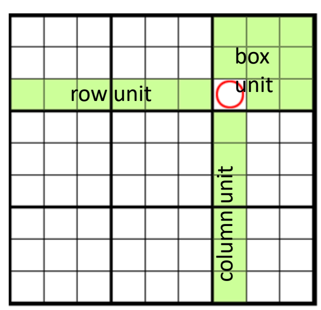
X domain [25, 100] $\quad X+Y \geq 200 \quad$ [75, 100]
Y domain [50, 125] $\quad\quad\quad\quad\quad\quad$ [100, 125]

How did we get [75, 100]? $Y = 125 \rightarrow X \geq 75$

35

Sudoku

- Puzzle game played with digit symbols
- All-different constraints exist on *units*
- Some cells initially filled in
- Hard for humans, pretty simple for CSP solvers



Jef Vandenberghe Sodoku tutorial

36

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		

37

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		

Sample constraints

- Alldiff(A1,A2,A3,A4,A5,A6,A7,A8,A9)
- Alldiff(A1,B1,C1,D1,E1,F1,G1,H1,I1)
- Alldiff(A1,A2,A3,B1,B2,B3,C1,C2,C3)

These can be expanded to binary constraints, e.g. A1≠A2

38

Sudoku

AC-3 constraint propagation

- E6: d={1, 2, 3, 4, 5, 6, 7, 8, 9}
- Box constraints:
  d_1 = d - {1, 2, 7, 8} = {3, 4, 5, 6, 8}
- Column constraints:
  d_2 = d_1 – {2, 3, 5, 6, 8, 9} = {4}

Therefore E6=4

	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		

39

Sudoku

AC-3 constraint propagation

- I6: d={1, 2, 3, 4, 5, 6, 7, 8, 9}
- Column constraints:
  d_1 = d - {2, 3, 4, 5, 6, 8, 9} = {1, 7}
- Row constraints:
  d_2 = d_1 – {1, 3, 5} = {7}

Therefore I6=7

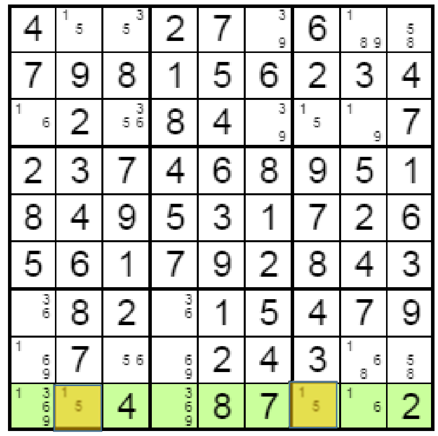
For this puzzle, continued application of AC-3 would solve the puzzle (not always true)

	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					4			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

40

Naked sets

- Yellow squares form a *naked pair* {1, 5}
  - one must contain1
  - other 5
- Can subtract 1 and 5 from domains of all other cells in row unit.
- These types of “tricks” are not limited to Sodoku puzzles.



41

Back to searching…

- Once all constraints have been propagated, search for a solution.
- Naïve search
  - Action picks a variable and a value. n variables domain size d → nd possible search nodes
  - Search on next variable.
  - Backtrack when search fails.
- Problems with naïve search
  - n variables with domains of size d
  - nd choices for first variable, (n-1)d for second….

$$nd \cdot (n-1)d \cdot \ldots \cdot 2d \cdot 1d = n!d^n$$

leaves but there are only d^n possible assignments!

42

Back to searching

- CSPs are commutative
- Order of variable selection does not affect correctness (may have other impacts)
- Modified search
  - Each level of search handles a specific variable.
  - Levels have d choices, leaving us with d^n leaves

43

Backtracking Search

```
def backtracking-search(CSP):
  return backtrack({}, CSP);  # call w/ no assignments

def backtrack(assignment, CSP):
  if all variables assigned, return assignment
  var = select-unassigned-variable(CSP, assignment)
  for each value in order-domain-values(var, assignment, csp):
    if value consistent with assignment:
      assignment.add({var = value})
      # propagate new constraints (optional)
      inferences = inference(CSP, var, assignment)
      if inferences ≠ failure:
        assignment.add(inferences)
        result = backtrack(assignment, CSP)
        if result ≠ failure, return result
    # either value inconsistent or further exploration failed
    # restore assignment to its state at top of loop and try next value
    assignment.remove({var = value}, inferences)
  # No value was consistent with the constraints
  return failure
```

44

Backtracking search

- Several strategies have been employed so far to make searches more efficient, e.g.
  - heuristics (best-first and A* search)
  - pruning (alpha-beta search)
- Can we come up with strategies to improve CSP search?

45

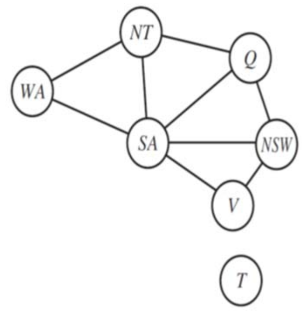
select-unassigned-variable

- Could try in order: $\{X_1, X_2, \ldots, X_n\}$
  Rarely efficient…
- Fail-first strategies
  - Minimum remaining value heuristic:
    Select the most constrained value; the one with the smallest domain.
    Rationale – probably the most likely variable to fail
  - Degree heuristic:
    Use the variable with the highest number of constraints on other unassigned variables.

46

select-unassigned-variable

- Minimum value remaining usually is a better performer, but not always:



All variables have domains of size three at start, but degree of constraints differs.

47

order-domain-values

- The order of the values within a domain may or may not make a difference
- Order has no consequence
  - if goal is to produce all solutions or
  - if there are no solutions
- In other cases, we use a fail-last strategy
  - Pick the value that reduces neighbors' domains as little as possible.

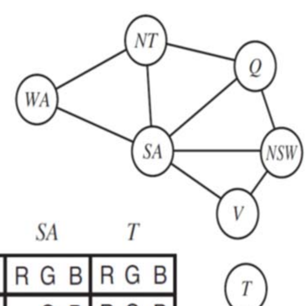
Why fail-first for variable selection and fail-last for value selection?

48

inference in search

- forward-checking
  - Check arc consistency with neighboring variables.
  - Not needed if arc-consistency was performed prior to search.

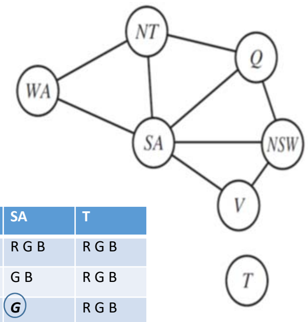
forward-checking example



	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After WA=red	R	G B	R G B	R G B	R G B	G B	R G B
After Q=green	R	B	G	R B	R G B	B	R G B
After V=blue	R	B	G	R	B		R G B

Note: Variable selection is not by degree ordering or min. remaining value

forward-checking example



with minimum remaining value heuristic

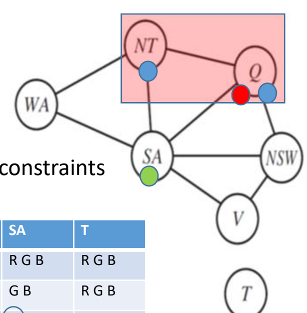
	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After WA=R	***R***	G B	R G B	R G B	R G B	G B	R G B
After SA=G	R	B	R B	R B	R B	***G***	R G B
After Q=R	R	B	***R***	B	R B	G	R G B
After V=R	R	B	R	B	***R***	G	R G B

51

forward-checking example

When we assigned SA=G, we restricted NT to B
However, Q was only restricted to R B

Arc-consistency does not check anything other than constraints with the neighbor being assigned.

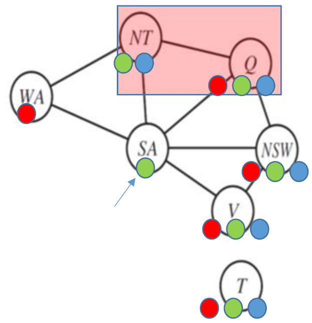


	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After WA=R	***R***	G B	R G B	R G B	R G B	G B	R G B
After SA=G	R	B	R B	R B	R B	***G***	R G B
After Q=R	R	B	***R***	B	R B	G	R G B
After V=R	R	B	R	B	***R***	G	R G B

52

Maintaining arc consistency (MAC)

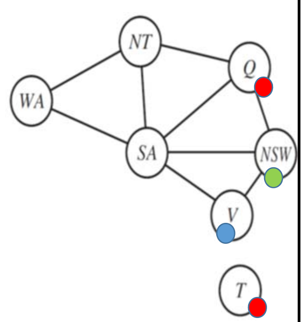
- Algorithm that propagates constraints beyond the node.
- AC3 algorithm with modified initial queue
  - typical AC3 – *all constraints*
  - MAC – constraints between selected variable and its neighbors



SA=green
queue: (SA, NT), (SA, Q),
(SA, NSW), (SA, V)

As (SA,NT) is processed and set to B, a constraint will be queued for (NT,Q). Between constraints on (NT,Q) and (SA,Q), Q will be resolved to R.

Intelligent backtracking



- Suppose variable ordering: Q, NSW, V, T, SA, WA, NT
- and assignments: {Q=red, NSW=green, V=blue, T=red}
- SA is problematic…
  - backtracking will try new values for Tasmania
- What if we could *backjump* to the variable that caused the problem?

Backjumping

- Maintain a *conflict set* for each variable X:
  A set of assignments that restricted values in X's domain.
- When a conflict occurs, we backtrack to the last conflict that was added.
- In the case of SA,
  - assignments to Q, NSW, and V restricted SA's domain
  - variable ordering: Q, NSW, V, T, SA, WA, NT
  - so we backjump to assignment of V with {Q=red,NSW=green}

last conflict added

55

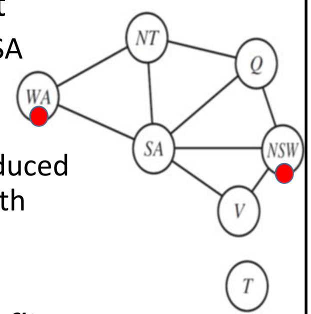
Backjumping implementation

- On forward checks of X assigned to x,
  - when X deletes a value from Y's domain, add X=x to Y's conflict set
  - If Y is emptied, add Y's conflict set to Xs and backjump
- Easy to implement, build conflict set during forward check.
- However, what we prune is redundant to what we'd prune from forward checking or MAC searches

56

More sophisticated backjumps…

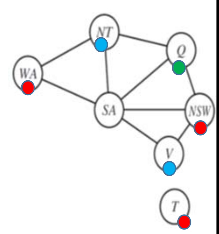
- Assignments to right are inconsistent
  - Suppose we try and assign T, NT, Q, V, SA
  - SA, NT, Q have reduced domains {green, blue} and cannot be assigned
  - Backjumping fails when a domain is reduced to ∅ as SA, NT, and Q are consistent with WA, NSW.



- Can we determine that there is a *conflict set* {WA, SA, NT, Q} that are causing the issue?

57

Conflict-directed backjumps



- Variable order: WA, NSW, T, NT, Q, V, SA
- SA fails. conf(SA) = {WA=red, NT=blue, Q=green}
- Last variable in conf(SA) is Queensland
  - Absorb SA's conflict set into Q
    $conf\,(Q) = conf\,(Q) \cup conf\,(SA) - \{Q\}$
  - conf(Q)
    = {NT=blue, NSW=red} U {WA=red, NT=blue,Q=green}-{Q=green}
    = {WA=red, NSW=red, NT=blue}
    Unable to assign a different color to Q, backjump
  - conf(NT) = conf(NT) U conf(Q)-{NT}
    = {WA=red} U {WA=red, NSW=red, NT=blue}
    = {WA=red, NSW=red}

Note: conf(SA) would have had NSW=red if NSW was processed before WA

58

Constraint-learning and no-goods

- On the Australias CSP, we identified a minimal set of assignments that caused the problem.
- We call these assignment *no-goods.*

- We can avoid running into this problem again by adding a new constraint (or checking a no-good cache).

59

Local Search CSPs

- Alternative to what we have seen so far
- Assign everything at once
- Search changes one variable at a time
  - Which variable?

60

Min-Conflicts Local Search

```
def minconflicts(csp, maxsteps):
  current = assign all variables
  for i = 1 to maxsteps:
    if solution(current), return current
    var = select conflicted variable at random from current
    val = find value that minimizes the number of conflicts
    update current such that var=val
  return failure
```

61

Min-Conflicts local search

- Pretty effective for many problems, e.g.
  million queens problem can be solved in about 50 steps
- This is essentially a greedy search, consequently:
  - local extrema
  - can plateau
  - many techniques discussed for hill climbing can be applied (e.g. simulated annealing, plateau search)

62

Structure of CSP problems

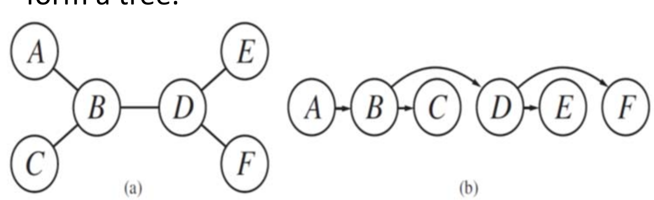
- Can we improve search by exploiting structure?
- Absolutely
  - Independent subproblems – solve separately
  - Tree structured CSP
    - Standard CSP: O(d^n) (domain size$^{\text{n variables}}$)
    - Given subproblems with *c* variables, we can solve in O($d^c n/c$)

This will not be on the exam.

63

Tree Structured CSP

- Basic ideas
  - Order variables (topological sort) such that constraints form a tree.



  - Solve one variable at a time, propagate

This will not be on the exam.

64

Tree Structured CSP

- Not all CSP constraints form trees.

- Transforming graphs with cycles into trees
  - Solve a variable that reduces the remaining conditions to a tree (e.g. South Australia node)
    or
  - Select a set of variables, a *cutset*, that reduce the problem to a tree after removal and examine problem with each possible assignment to the cutset.

This will not be on the exam.