22c:145 Artificial Intelligence

Constraint Satisfaction Problems (CSP)

Chapter 6, Textbook

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 ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

Example: Map-Coloring

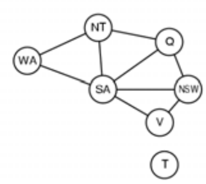
Solutions are complete and consistent assignments, e.g., WA = red, NT = green, Q = red,NSW = green,V = red,SA = blue,T = green

(Aside: Four colors suffice. Appel and Haken 1977)

Constraint graph: Graph Coloring

Binary CSP: each constraint relates two variables

Constraint graph: nodes are variables, arcs are constraints

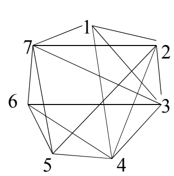


Two variables are adjacent or neighbors if they are connected by an edge or an arc

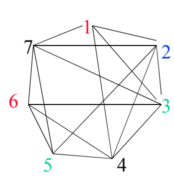
Application of Graph Coloring

Lots of applications involving scheduling and assignments.

Scheduling of final exams – nodes represent finals, edges between finals denote that both finals have common students (and therefore they have to have different colors, or different periods).



Graph of finals for 7 courses



Time Period → courses

I (red) → 1,6
II (blue)→2
III (green)→3,5
IV (black)→

CSP: Constraint Satisfaction Problems

Set of vars, set of possible values for each vars & set of constraints defines a CSP.

A solution to the CSP is an assignment of values to the variables so that all constraints are satisfied (no "violated constraints.")

A CSP is inconsistent if no such solution exists.
Eg try to place 9 non-attacking queens on an 8x8 board.

Constraint Satisfaction Problem

Set of variables $\{X_1, X_2, \ldots, X_n\}$

Each variable X_i has a domain D_i of possible values

Usually D_i is discrete and finite

Set of constraints $\{C_1, C_2, \ldots, C_p\}$

Each constraint C_k involves a subset of variables and specifies the allowable combinations of values of these variables

Goal:

Assign a value to every variable such that all constraints are satisfied

7

Varieties of CSPs

Discrete variables

our focus

- **finite domains:**
  - ***n* variables, domain size *d* → $O(d^n)$ complete assignments**

  (includes Boolean satisfiability 1st problem which is known 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., $StartJob_1 + 5 \leq StartJob_3$

Continuous variables

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

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., cryptarithmetic column constraints**

9

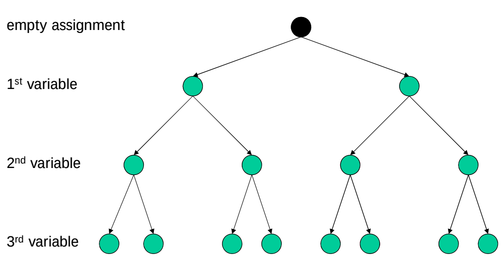
Solving CSP by search : Backtrack Search

BFS vs. DFS

- **BFS → not a good idea.**
  - **Reduction by commutativity of CSP**
    - **A solution is not in the permutations but in combinations.**
    - **A tree with d^n leaves**
- **DFS**
  - **Used popularly**
    - **Every solution must be a complete assignment and therefore appears at depth *n* if there are *n* variables**
    - **The search tree extends only to depth n.**
  - **A variant of DFS: Backtrack search**
    - **Chooses values for one variable at a time**
    - **Backtracks when failed *even before reaching a leaf.***
  - ***Better than BFS due to backtracking but still need more "cleverness" (reasoning/propagation).***

11

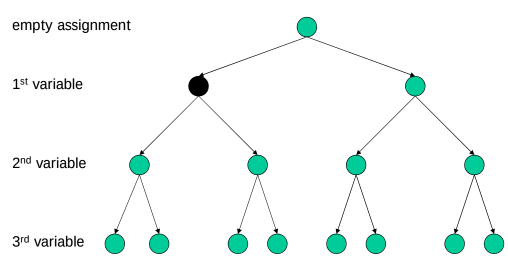
→ Backtrack Search



Assignment = {}

13

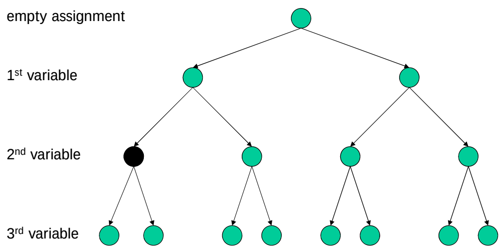
→ Backtrack Search



Assignment = {(var1=v11)}

14

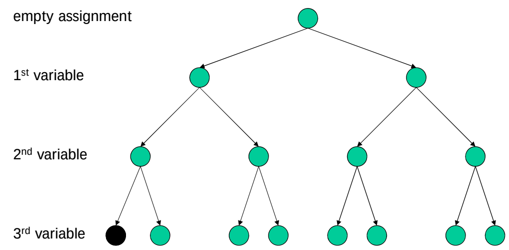
→ Backtrack Search



Assignment = {(var1=v11),(var2=v21)}

15

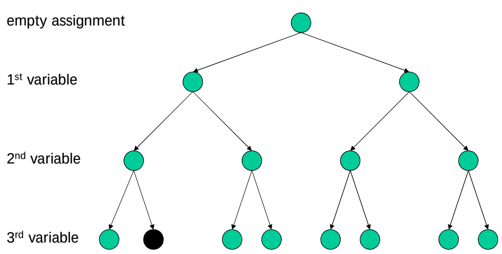
→ Backtrack Search



Assignment = {(var1=v11),(var2=v21),(var3=v31)}

16

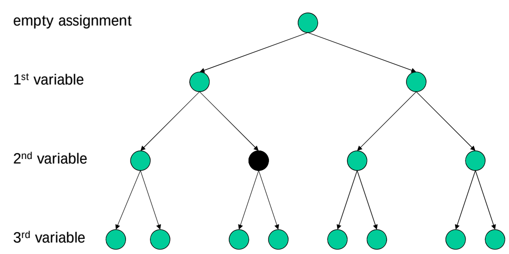
→ Backtrack Search



Assignment = {(var1=v11),(var2=v21),(var3=v32)}

17

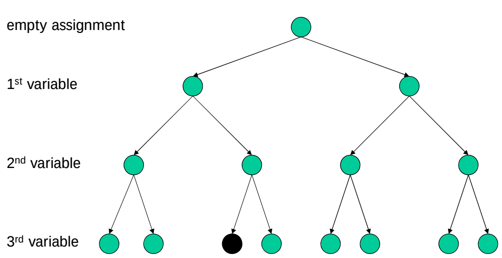
→ Backtrack Search



Assignment = {(var1=v11),(var2=v22)}

18

→ Backtrack Search



Assignment = {(var1=v11),(var2=v22),(var3=v31)}

19

CSPs vs Search Problems

States and goal test have a *standard* representation.

- state is defined by variables X_i with values from domain D_i
- goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Interesting tradeoff:

Constraints can use a formal representation language.

Allows useful general-purpose algorithms more powerful than standard search algorithms that have to resort to problem specific heuristics to enable solution of large problems.

21

Key issue: For search problems, we have treated nodes in search trees as "black boxes," only looked inside to check its heuristic value or whether the node is a goal state.

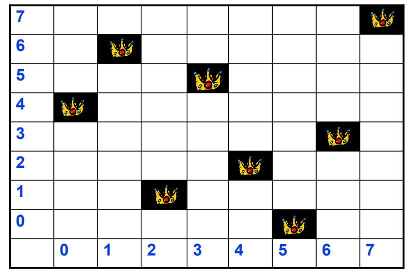
In CSPs, we want to "look inside the nodes" and exploit problem structure during the search. Sometimes, reasoning or inference ("propagation techniques") will led us find solutions without any search!

22

N-Queens

The standard N by N Queen's problem asks how to place N queens on an ordinary chess board so that they don't attack each other

N=8



23

How do we search for a solution?

Start with empty variable assignment (no vars assigned). Then, build up partial assignments until all vars assigned.

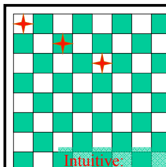
Action: "assign a variable a value."
Goal test: "all vars assigned and no constraint violation."

What is the search space? (n vars, each with d possible values)

Top level branching: n . d
Next branching: (n-1) . d
Next branching: (n-2) . d
…
Bottom level: d

"Only" n^d distinct value assignments!
Different var ordering can lead to the same assignment! Wasteful…
Just "fix" a variable ordering:
Backtrack search.
Check only n^d full var-value assignments.

24



Backtrack search

Aside: "legal" and "feasible"
Already assumes a bit of "reasoning." (Next.)

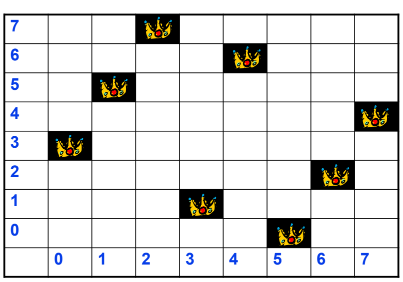
There are many improvements on intuitive idea…

Intuitive:
1) Fix some ordering on the variables. Eg x1, x2, x3 …
2) Fix some ordering on possible values. Eg 1, 2, 3, …
3) Assign first variable, first (legal) value.
4) Assign next variable, its first (legal) value.
5) Etc.
6) Until no remaining (feasible) value exist for variable x_i, backtrack to previous var setting of x_(i-1), try next possible setting. If none exists, move back another level. Etc.

Visually, very intuitive on the N-Queens board ("the obvious strategy").
See figure 6.5 book. Recursive implementation. Practice: Iterative with stack.

25

N-Queens
N=8 (another solution)



26

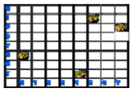
Can a search program figure out that you can't place 101 queens on 100x100 board?

Not so easy! Most search approaches can't. *Need much more clever reasoning, instead of just search. (Need to use Pigeon Hole principle.)*

Aside: Factored representation does not even allow one to ask the question. Knowledge is build in.)

Alternative question: With N queens is there a solution with queen in bottom right corner on a N x N board?

27



Partially Filled N-queens

So the N-queens problem is easy when we start with an empty board.

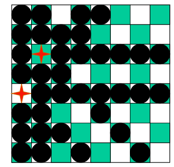
What about if we pre-assign some queens and ask for a completion?

Similar Problem: Sudoku Puzzles

28

Reasoning, inference or "propagation."

Message:
CSP propagation techniques can dramatically reduce search.
Sometimes to no search at all! Eg. Sudoku puzzles.



After placing the first queen, what would you do for the 2nd?

29

General Search vs. Constraint satisfaction problems (CSPs)

Standard search problem:

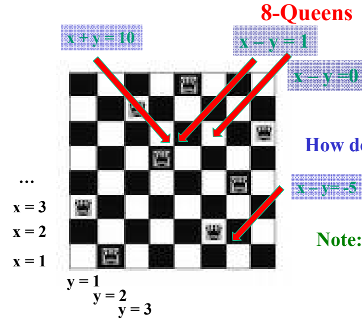
– state is a "black box" – can be accessed only in limited way: successor function; heuristic function; and goal test.

What is needed for CSP:
Not just a successor function and goal test. Also a means of propagating the constraints (e.g. imposed by one queen on the others and an early failure test).

→ Explicit and formal representation of constraints and constraint manipulation algorithms

30

Motivational Example: 8-Queens



How do we represent 8-Queens as a CSP:
Variables?
Constraints?

Note: More than one option.

31

8-Queens Problem as CSP

Xi: column for queen in row i

8 variables X_i, i = 1 to 8 (one per row)

Domain for each variable {1,2,…,8}

Constraints are of the form:

– $X_i \neq X_j$ when $j \neq i$ (i.e. no two in the same column)

– **No queens in same diagonal:**

1) $X_i - X_j \neq i - j$

2) $X_i - X_j \neq j - i$

(check that this works!)

Alternative? **Boolean vars**

32

Boolean Encoding for 8-Queen

64 Boolean variables X_{ij}, i = 1 to 8, j = 1 to 8

Domain for each variable {0,1} (or {False, True})

Constraints are of the form:

X_{ij} = 1 iff "there is a queen on location (i,j)."

Row and columns

- **If ($X_{ij} = 1$) then ($X_{ik} = 0$) for all k = 1 to 8, k≠j (logical constraint)**
- **$X_{ij} = 1 \Rightarrow X_{kj} = 0$ for all k = 1 to 8, k≠i**

Diagonals

- **$X_{ij} = 1 \Rightarrow X_{i+l,j+l} = 0$ l = 1 to 7, i+l ≤8; j+l≤8 (right and up)**
- **$X_{ij} = 1 \Rightarrow X_{i-l,j+l} = 0$ l = 1 to 7, i-l ≥1; j+l≤8 (right and down)**
- **$X_{ij} = 1 \Rightarrow X_{i-l,j-l} = 0$ l = 1 to 7, i-l ≥1; j-l≥1 (left and down)**
- **$X_{ij} = 1 \Rightarrow X_{i+l,j-l} = 0$ l = 1 to 7, i+l ≤8; j-l≥1 (left and up)**

What's missing? **Need N (= 8) queens on board!**

3 options:
1) Maximize sum X_ij (optimization formulation)
2) Sum X_ij = N (CSP; bit cumbersome in Boolean logic)
3) For each row i: (X_i1 OR X_i2 OR X_i3 … X_iN)

33

Logical equivalence

Two sentences **p** an **q** are logically equivalent (≡ or ⇔) iff p ↔ q is a tautology (and therefore p and q have the same truth value for all truth assignments)

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of $\vee$
$\neg(\neg\alpha) \equiv \alpha$ double-negation elimination
$(\alpha \to \beta) \equiv (\neg\beta \to \neg\alpha)$ contraposition
$\boxed{(\alpha \to \beta) \equiv (\neg\alpha \vee \beta) \text{ implication elimination}}$
$(\alpha \leftrightarrow \beta) \equiv ((\alpha \to \beta) \wedge (\beta \to \alpha))$ biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ de Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ de Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of $\vee$ over $\wedge$

34

SAT: Propositional Satisfiability problem

Satifiability (SAT): Given a formula in propositional calculus, is there a model (i.e., a satisfying interpretation, an assignment to its variables) making it true?

We consider clausal form, e.g.:

$(a \vee \neg b \vee \neg c) \wedge (b \vee \neg c) \wedge (a \vee c)$

2^n possible assignments

SAT: prototypical hard combinatorial search and reasoning problem. Problem is NP-Complete. (Cook 1971)
Surprising "power" of SAT for encoding computational problems.

35

Significant progress in Satisfiability Methods

Software and hardware verification – complete methods are critical - e.g. for verifying the correctness of chip design, using SAT encodings

Many Applications: Hardware and Software Verification Planning, Protocol Design, Scheduling, Materials Discovery etc.

Going from 50 variable, 200 constraints to 1,000,000 variables and 5,000,000 constraints in the last 10 years

Current methods can verify automatically the correctness of large portions of a chip

36

Bounded Model Checking instance:

The instance bmc-ibm-6.cnf, IBM LSU 1997:

```
p cnf 51639 368352
-1 7 0
-1 6 0
-1 5 0
-1 -4 0
-1 3 0
-1 2 0
-1 -8 0
-9 15 0
-9 14 0
-9 13 0
-9 -12 0
-9 11 0
-9 10 0
-9 -16 0
-17 23 0
-17 22 0
```

**_i.e._ ((not x_1) or x_7)
and ((not x_1) or x_6)
and … _etc._**

37

Dimacs Format for CNF

File format

The benchmark file format will be in a simplified version of the DIMACS format: c

c start with comments
c
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0

The file can start with comments, that is lines begining with the character c.

Right after the comments, there is the line p cnf nbvar nbclauses indicating that the instance is in CNF format; nbvar is the exact number of variables appearing in the file; nbclauses is the exact number of clauses contained in the file.

Then the clauses follow. Each clause is a sequence of distinct non-null numbers between -nbvar and nbvar ending with 0 on the same line; it cannot contain the opposite literals i and -i simultaneously. Positive numbers denote the corresponding variables. Negative numbers denote the negations of the corresponding variables.

38

SAT Instance for 4x4 Queen

```
p cnf 16 84
1 2 3 4  0
-1 -2  0
-1 -3  0
-1 -4  0
-2 -3  0
-2 -4  0
-3 -4  0
5 6 7 8  0
-5 -6  0
-5 -7  0
-5 -8  0
-6 -7  0
-6 -8  0
-7 -8  0
9 10 11 12  0
-9 -10  0
-9 -11  0
-9 -12  0
-10 -11  0
-10 -12  0
-11 -12  0
13 14 15 16  0
-13 -14  0
-13 -15  0
-13 -16  0
-14 -15  0
-14 -16  0
-15 -16  0
```

```
1 5 9 13  0
-1 -5  0
-1 -9  0
-1 -13  0
-5 -9  0
-5 -13  0
-9 -13  0
2 6 10 14  0
-2 -6  0
-2 -10  0
-2 -14  0
-6 -10  0
-6 -14  0
-10 -14  0
3 7 11 15  0
-3 -7  0
-3 -11  0
-3 -15  0
-7 -11  0
-7 -15  0
-11 -15  0
4 8 12 16  0
-4 -8  0
-4 -12  0
-4 -16  0
-8 -12  0
-8 -16  0
-12 -16  0
```

```
-1 -6  0
-1 -11  0
-1 -16  0
-6 -11  0
-6 -16  0
-11 -16  0
-2 -7  0
-2 -12  0
-7 -12  0
-3 -8  0
-5 -10  0
-5 -15  0
-10 -15  0
-9 -14  0
-4 -7  0
-4 -10  0
-4 -13  0
-7 -10  0
-7 -13  0
-10 -13  0
-3 -6  0
-3 -9  0
-6 -9  0
-2 -5  0
-8 -11  0
-8 -14  0
-11 -14  0
-12 -15  0
```

39

Which encoding is better?Allows for faster solutions?

One would think, fewer variables is better…

Search spaces:

$8^8 = 1.6 \times 10^6$ vs $2^{64} = 1.8 \times 10^{19}$

However, in practice SAT encodings can be surprisingly effective, even with millions of Boolean variables. Often, few true local minima in search space.

40

Remark

Finite CSP include 3SAT as a special case (under logical reasoning). 3SAT is known to be NP-complete.

So, in the worst-case, we cannot expect to solve a finite CSP in less than exponential time.

41

Example: Crypt-arithmetic Puzzle

```
  S E N D
+ MOR E
 ----------
 MONEY
```

Variables: S, E, N, D, M, O, R, Y

Domains:
[0..9] for S, M, E, N, D, O, R, Y

Search space: 1,814,400
Aside: could have [1..9] for S and M

```
Soln.:9567
      1085
      ====
     10652
```

42

Constraints

Option 1:
C1a) 1000 S + 100 E + 10 N + D +
1000 M + 100 O + 10 R + E
= 10000 M + 1000 O + 100 N + 10 E + Y
Or use 5 equality constraints, using auxiliary "carry" variables C1, …, C4 Є [0…9]

```
 SEND
+MORE
----------
MONEY
```

Option 2: C1b)
D + E = 10 C1 + Y
C1 + N + R = 10 C2 + E
C2 + E + O = 10 C3 + N
C3 + S + M = 10 C4 + O
C4 = M

Which constraint set better for solving? C1a or C1b? Why?

C1b, more "factored". Smaller pieces. Gives more propagation!

Need two more sets of constraints:

C2) S =/= 0, M =/= 0
C3) S =/= M, S =/= O, … E =/= Y (28 not equal constraints)

Note: need to assert everything!
Alt. "All_diff(S,M,O,...Y)" for C3.

43

Some Reflection: Reasoning/Inference vs. Search

How do human solve this?
What is the first step?

```
        S  E  N  D
+       M  O  R  E
 ------------------
=   M   O  N  E  Y
```

1) M = 1, because M =/= 0 and …
the carry over of the addition of two digits (plus previous carry) is at most 1.

Actually, a somewhat subtle piece of *mathematical background knowledge*.

Also, what made us focus on M?
Experience / intuition …

44

```
        S  E  N  D
+       M  O  R  E
 ------------------
=   M   O  N  E  Y
```

1) M = 1, because M =/= 0 and …
the carry over of the addition of two digits (plus previous carry) is at most 1.

2) O = 0. Because M=1 and we have to have a carry to the next column. S + 1 + C3 is either 10 or 11. So, O equals 0 or 1. 1 is taken. So, O = 0.

3) S = 9. There cannot be a carry to the 4th column (if there were, N would also have to be 0 or 1. Already taken.). So, S = 9.

A collection of "small pieces" of local reasoning, using basic constraints from the rules of arithmetic. A logic (SAT) based encoding will likely "get these steps."

45

And further it goes…

4. If there were no carry in column 3 then E = N, which is impossible. Therefore there is a carry and N = E + 1.
5. If there were no carry in column 2, then (N + R) mod 10 = E, and N = E + 1, so (E + 1 + R) mod 10 = E which means (1 + R) mod 10 = 0, so R = 9. But S = 9, so there must be a carry in column 2 so **R = 8**.
6. To produce a carry in column 2, we must have D + E = 10 + Y.
7. Y is at least 2 so D + E is at least 12.
8. The only two pairs of available numbers that sum to at least 12 are (5,7) and (6,7) so either E = 7 or D = 7.
9. Since N = E + 1, E can't be 7 because then N = 8 = R so **D = 7**.
10. E can't be 6 because then N = 7 = D so **E = 5** and **N = 6**.
11. D + E = 12 so **Y = 2**.

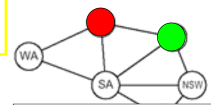
Largely, a clever chain of reasoning / inference / propagation steps (no search) except for… exploring 2 remaining options (i.e., search) to find complete solution.

46

Improving Backtracking Efficiency

- Which variable should be assigned next?
  - Minimum Remaining Values heuristic
- In what order should its values be tried?
  - Least Constraining Values heuristic
- Can we detect inevitable failure early?
  - Forward checking
  - Constraint propagation (Arc Consistency)
- When a path fails, can the search avoid repeating this failure?
  - Backjumping
- Can we take advantage of problem structure?
  - Tree-structured CSP

Variable & value ordering to increase the likelihood to success



Early failure-detection to decrease the likelihood to fail

Restructuring to reduce the problem's complexity

General purpose techniques

Improving backtracking efficiency

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

function RECURSIVE-BACKTRACKING(assignment, csp) returns 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 ← RECURSIVE-BACKTRACKING(assignment, csp)
       if result != failure then return result
       remove {var = value} from assignment
  return failure
```

Choice of Variable

#1: Minimum Remaining Values (aka Most-constrained-variable heuristic):

Select a variable with the fewest remaining values

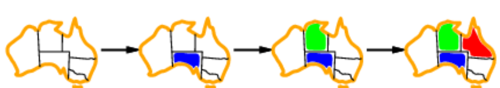
49

Choice of Variable, cont.

Tie-breaker among most constrained variables

#2 Most constraining variable:

- **choose the variable with the most constraints on remaining variables**
-

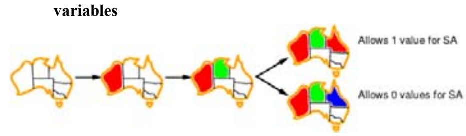


50

Choice of Value: Least constraining value

#3 Given a variable, choose the least constraining value:

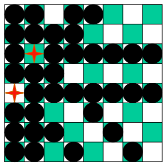
- **the one that rules out the fewest values in the remaining variables**



51

Constraint Propagation

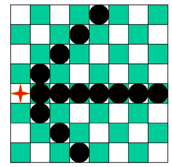
The process of determining how the possible values of one variable affect the possible values of other variables



52

Forward Checking

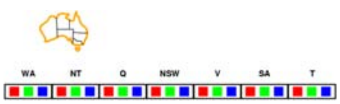
After a variable X is assigned a value v, look at each unassigned variable Y that is connected to X by a constraint and deletes from Y's domain any value that is inconsistent with v



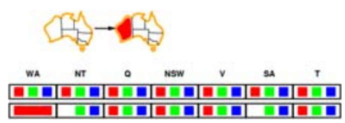
Terminate branch when any variable has no legal values & backtrack.

53

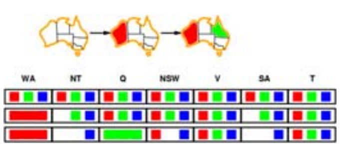
Forward checking



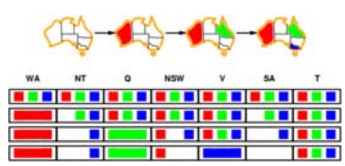
54



55

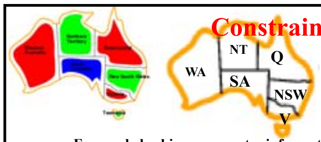


56



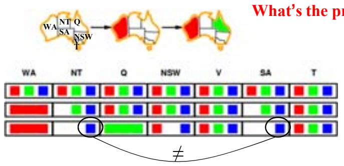
57

Constraint propagation



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

What's the problem here?



NT and SA cannot both be blue!

Use: constraint propagation ***repeatedly to enforce constraints locally***.

Definition (Arc consistency)

A constraint C_xy is said to be arc consistent w.r.t. x iff
for each value v of x there is an allowed value of y.

Similarly, we define that C_xy is arc consistent w.r.t. y.

A binary CSP is arc consistent iff every constraint C_xy is arc consistent wrt x as well as wrt y.

When a CSP is not arc consistent, we can make it arc consistent.

This is also called "enforcing arc consistency".

Example

Let domains be
D_x = {1, 2, 3}, D_y = {3, 4, 5, 6}
One constraint
C_xy = {(1,3), (1,5), (3,3), (3,6)} ["allowed value pairs"]

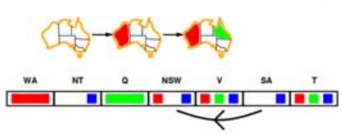
C_xy is not arc consistent w.r.t. x, neither w.r.t. y. Why?

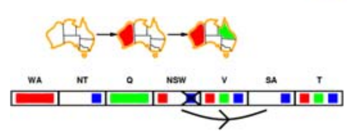
To enforce arc consistency, we filter the domains, ***removing*** **inconsistent values.**

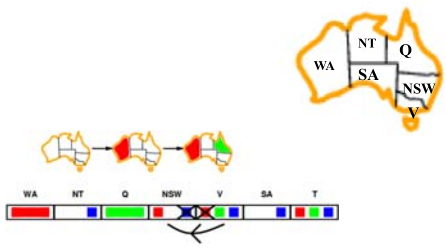
D'_x = {1, 3}, D'_y={3, 5, 6}

Arc consistency

Simplest form of propagation makes each arc consistent.
I.e., X → *Y* is consistent iff
for every value *x* of *X* there is some allowed *y*

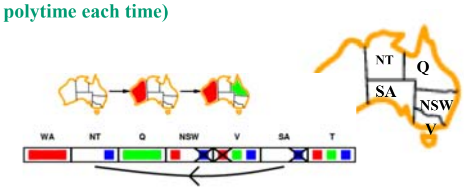






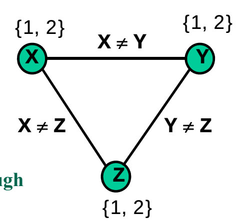
If *X* loses a value, neighbors of *X* need to be rechecked.
Arc consistency detects failure earlier than forward checking.

Can be run as a preprocessor or after each assignment.
(takes polytime each time)



Empty domain detected! Backtrack early.

Beyond Arc Consistency



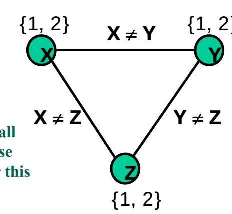
Is this network arc consistent?

What is the solution?

Clearly arc consistency is not enough to guarantee global consistency. There are other forms of consistency, such as k-consistency.

But when k = n (num vars), we are looking at the original problem!

k - Consistency



A graph is K-consistent iff the following is true:

Choose values of any K-1 variables that satisfy all the constraints among these variables and choose any K th variable. Then, there exists a value for this K th variable that satisfies all the constraints among these K variables.

A graph is strongly K-consistent if it is J-consistsent for all J<=K.

What type of consistency would we need here to solve any constraint problem without search?
K = N

Consistency

Node consistency = strong 1- consistency
Arc consistency = strong 2- consistency
(note: arc-consistency is usually assumed to include node-consistency as well).
See Textbook sect. 6.2.3 for "path-consistency" = 3-consistency for binary CSPs.

Algorithms exist for making a constraint graph strongly K-consistent for K>2 but in practice they are rarely used because of efficiency issues.

Other consistency notions involve "global constraints," spanning many variables. E.g. AllDiff constraint can handle Pigeon Hole principle.

Summary: Solving a CSP

Search:
- **can find solutions, but may examine many non-solutions along the way**

Constraint Propagation:
- **can rule out non-solutions, but but may not lead to full solution.**

Interweave constraint propagation and search
- **Perform constraint propagation at each search step.**
- ***Goal: Find the right balance between search (backtracking) and propagation (reasoning).***

Surprising efficiency (last 10 yrs):
100K + up to one million variable CSP problems are now solvable!

See also local search. Textbook 6.4