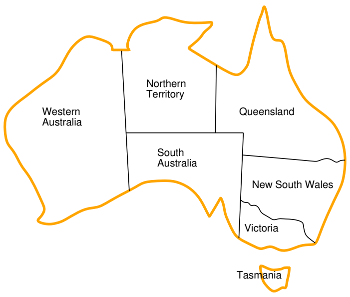
Constraint Satisfaction Problems

- Intro CSP
  Def'n, Examples, Types
- Naive Generate-&-Test
- Complexity
- Improvements to Generate-&-Test
- Special Case:
  Tree structured constraints
- Constraint *Optimization* Problems

Example: Map Coloring



- **Variables**: $\{V, T, WA, NT, SA, Q, NSW\}$
- **Domains**: $\mathcal{D}_i = \{\mathtt{r}, \mathtt{g}, \mathtt{b}\}$
  (same domain for each)
- **Constraints**: No adjacent "countries" can have same color

  $C_{WA,NT}$ constrains values for WA and NT:
  $\ldots C_{WA,\ NT} = \{ \langle \mathtt{r}, \mathtt{g} \rangle, \langle \mathtt{r}, \mathtt{b} \rangle, \langle \mathtt{g}, \mathtt{r} \rangle, \langle \mathtt{g}, \mathtt{b} \rangle, \langle \mathtt{b}, \mathtt{r} \rangle, \langle \mathtt{b}, \mathtt{g} \rangle \}$
- Similarly:
  $C_{WA,NT},\ C_{WA,SA},\ C_{NT,SA},\ C_{NT,Q},$
  $C_{SA,Q},\ C_{SA,NSW},\ C_{SA,V},\ C_{Q,NSW},\ C_{NSW,V}$

Formal Def'n of CSP

- **Constraint Satisfaction Problem (CSP)**:

$\mathcal{X}$	$=$	$\{X_1, X_2, \ldots, X_n\}$	set of variables
each with finite domain $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$.			
$\mathcal{C}$	$=$	$\{C_1, C_2, \ldots, C_m\}$	set of constraints

- Each **constraint** C_i:
  - deals with subset of variables $\{X_1, X_7, X_8\}$
  - restricts values assigned to those var's

  (specifies which value-sets are *compatible* for those variables)
- A **solution** is
  assignment of values to *all* variables
  that satisfies *all* constraints.

Example:

```
  SEND
+ MORE
 MONEY
```

Find subst. of digits for letters s.t. sum is correct.
(Each letter stands for different digit.)

- **Variables:** $\{S, E, N, D, M, O, R, Y\}$
- **Domains:** $D_i = \{0, \ldots, 9\}\ \forall i$
- **Constraints#1:**
  $(1000 \times S + 100 \times E + 10 \times N + D)$
  $+\ (1000 \times M + 100 \times O + 10 \times R + E)$
  $=\ (10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y)$

  Unique: each letter is different
  $S \neq E, \quad S \neq N, \quad \ldots$
- **Constraints#2:**
  $$\begin{aligned} D + E &= Y + 10 \times c_1 \\ N + R + c_1 &= E + 10 \times c_2 \\ E + O + c_2 &= N + 10 \times c_3 \\ S + M + c_3 &= O + 10 \times M \end{aligned}$$
  (c_i is "carry")

  $+$ Unique: each letter is different …

Examples

- Assignment problems
  ...who teaches what class
- Time-tabling problems
  ...which class is offered when & where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floor-planning

- Map-coloring
  Crypto-arithmetic

Constraint Satisfaction/Optimization Problems

- Scheduling Courses:
  Assign time/prof/room to each course
  - "Hard Constraints" (requirements)
    + Prof can only be at one place at any time
    + Course + Lab must be at different times
    + Only one course to a room, ...
  - "Soft Constraints" (preferences)
    + Companion classes should be close in time
    + Avoid 8am
    + Minimize total number of rooms used...

  + scheduling maintanence, equipment usage, ...
- VLSI Layout:
  Find position for various subparts
  - "Hard Constraints"
    + Achieve certain functionality
    + Upper bound on clock-cycle time
  - "Soft Constraints"
    + Minimize region
    + Minimize wire-length
    + Minimize congestion, ...

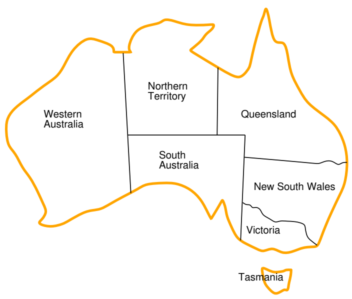
  + part assembly, ...

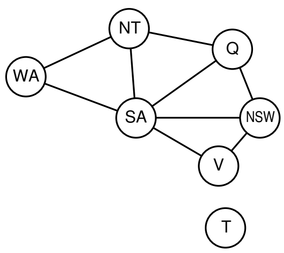
Very General Formalism

- **Multi-dimensional Selection Problems**
  Given set of variables
  each w/ domain (set of possible values)
  assign a value to each variable that either

  1. satisfies given set of "hard" constraints:
     satisfiability problems
     or
  2. minimizes given cost function, where each assignment to variables has cost:
     optimization problems — "soft constraints"
- In general,
  + different domains for diff var's
  (discrete, or continuous $X + Y > Z + 3$)
  + different constraints for diff var-tuples
  + constraints over k-tuples of vars ($k > 2$)
- Our focus:
  Any feasible solution, Hard constraints

Australian MapColor Constraint Network





Prolog Encoding — Australia MC

```
   %%% Domain information
domWA(A) :- color(A).
domNT(B) :- color(B).
domSA(C) :- color(C).
domQ(D) :- color(D).
domNSW(E) :- color(E).
domV(F) :- color(F).
domT(F) :- color(F).

color(r).    color(g).    color(b).  % color(X) :- member(X, [r,g,b] ).

  %%% Constraint information
cWA_NT(WA,NT) :- diff(WA,NT).    cWA_SA(WA,SA) :- diff(WA,SA).
cNT_SA(NT,SA) :- diff(NT,SA).    cNT_Q(NT,Q) :- diff(NT,Q).
cSA_Q(SA,Q) :- diff(SA,Q).       cSA_NSW(SA,NSW) :- diff(SA,NSW).
cSA_V(SA,V) :- diff(SA,V).
cQ_NSW(Q,NSW) :- diff(Q,NSW).    cNSW_V(NSW,V) :- diff(NSW,V).

 % diff(A,B) :- A /== B.
diff(r,g).  diff(g,r).  diff(b,r).  diff(r,b).  diff(g,b).  diff(b,g).

  %%% Complete problem
answer([WA,NT,SA,Q,NSW,V,T]) :-  domWA(WA), domNT(NT),  domSA(SA),
   domQ(Q), domNSW(NSW),  domV(V), domT(T),
   cWA_NT(WA,NT), cWA_SA(WA,SA), cNT_SA(NT,SA), cNT_Q(NT,Q), cSA_Q(SA,Q),
   cSA_NSW(SA,NSW), cQ-NSW(Q,NSW), cNSW_V(NSW,V), cSA_V(SA,V).

| ? - bagof( X, answer(X), S).
S = [[r,g,b,r,g,r,r], [r,g,b,r,g,r,g], [r,g,b,r,g,r,b], [r,b,g,r,b,r,r],
    [r,b,g,r,b,r|...], [r,b,g,r,b|...], [g,r,b,g|...], [g,r,b|...],
    [g,r|...], [g|...]|...] ? ;
no
| ? -
```


Prolog Encoding: SEND+MORE=MONEY

```
  %%% Domain information
domS(S) :- pdigit(S).    domE(E) :- digit(E).
domN(N) :-  digit(N).    domD(D) :- digit(D).
domM(M) :- pdigit(M).    domO(O) :- digit(O).
domR(R) :-  digit(R).    domY(Y) :- digit(Y).
domC1(C1) :- v01(C1).   domC2(C2) :- v01(C2).
domC3(C3) :- v01(C3).

digit(X) :- pdigit(X).
digit(0).
            pdigit(1).  pdigit(2).  pdigit(3).  pdigit(4).
pdigit(5).  pdigit(6).  pdigit(7).  pdigit(8).  pdigit(9).
v01(0).  v01(1).

  %%% Constraint information
  % each pair is different
cSE(S,E) :- diff(S,E).   cSN(S,N) :- diff(S,N).  cSD(S,D) :- diff(S,D).
cSM(S,M) :- diff(S,M).   cSO(S,O) :- diff(S,O).  cSR(S,R) :- diff(S,R).
cSY(S,Y) :- diff(S,Y).

cEN(E,N) :- diff(E,N).   cED(E,D) :- diff(E,D).  cEM(E,M) :- diff(E,M).
cEO(E,O) :- diff(E,O).   cER(E,R) :- diff(E,R).  cEY(E,Y) :- diff(E,Y).

cND(N,D) :- diff(N,D).   cNM(N,M) :- diff(N,M).  cNO(N,O) :- diff(N,O).
cNR(N,R) :- diff(N,R).   cNY(N,Y) :- diff(N,Y).

cDM(D,M) :- diff(D,M).   cDO(D,O) :- diff(D,O).  cDR(D,R) :- diff(D,R).
cDY(D,Y) :- diff(D,Y).

cMO(M,O) :- diff(M,O).   cMR(M,R) :- diff(M,R).  cMY(M,Y) :- diff(M,Y).

cOR(O,R) :- diff(O,R).    cOY(O,Y) :- diff(O,Y).

cRY(R,Y) :- diff(R,Y).
```



```
diff(X,Y) :- X \== Y.
 % or could have all 55 pairs...

 % ''addition'':
cDEY1(D,E,Y,C1) :- 0 is ((D+E) - (Y + 10*C1)).
cNR1E2(N,R,C1,E,C2) :- 0 is ((N+R+C1) - (E + 10*C2)).
cEO2N3(E,O,C2,N,C3) :- 0 is ((E+O+C2) - (N + 10*C3)).
cSM3OM(S,M,C3,O,M) :- 0 is ((S+M+C3) - (O + 10*M)).

  %%% Total constraint:
ans([S,E,N,D,M,O,R,Y]) :-
  domS(S), domE(E), domN(N), domD(D), domM(M),
  domO(O), domR(R), domY(Y), domC1(C1), domC2(C2), domC3(C3),

  cSE(S,E), cSN(S,N), cSD(S,D), cSM(S,M), cSO(S,O), cSR(S,R), cSY(S,Y),
            cEN(E,N), cED(E,D), cEM(E,M), cEO(E,O), cER(E,R), cEY(E,Y),
                      cND(N,D), cNM(N,M), cNO(N,O), cNR(N,R), cNY(N,Y),
                                cDM(D,M), cDO(D,O), cDR(D,R), cDY(D,Y),
                                          cMO(M,O), cMR(M,R), cMY(M,Y),
                                                    cOR(O,R), cOY(O,Y),
                                                              cRY(R,Y),
  cDEY1(D,E,Y,C1), cNR1E2(N,R,C1,E,C2), cEO2N3(E,O,C2,N,C3),
  cSM3OM(S,M,C3,O,M).
```

Complexity of CSP

- Propositional Satisfiability is CSProblem.

  (Domain of each variable: $\{\mathtt{t}, \mathtt{f}\}$
  Each k-clause allows $2^k - 1$ assignments, ...)

⇒ **Every NP-complete** problem can be formulated as CSProblem.

  ...so CSPs are HARD to solve!

- Approaches
  - Seek algorithms that work well on typical cases even though worst case may be exponential
  - Seek special cases w/ efficient algorithms
  - ∘ Seek efficient approximation algorithms
  - ∘ Develop parallel / distributed algorithms

Search Approach to CSP

- "Grow"
  - **State**: partial assignment
    Initial state = $\langle\rangle$
  - **Operators**:
    1. Assign value to *any* unassigned variable
       (Branching Factor: $\sum_i |\mathcal{D}_i|$)
    2. Assign value to $k+1^{st}$ variable
       (Branching Factor: $\max_i |\mathcal{D}_i|$)
- "Modify/Repair"
  - **State**: *complete* assignment
    Initial state: random(?)
  - **Operator**: Change value of some variable

\+ . . . in all cases:
  - **Goal-test**: all variables asssigned,
    all constraints satisfied
  - **PathCost**: 0

Note: Goal test is DECOMPOSED into
individual constraints!

If $A \neq B$, then
$\langle A = 1,\ B = 1,\ \ldots\rangle$ cannot be part of solution. . .
$\Rightarrow$ can be pruned!

"Modify/Repair" Approach: Exhaustive

Initial State: all variables are assigned

Operators: re-assign new value to variable

Goal test: all constraints are satisfied

- aka **Generate-and-Test Algorithm**

  Sequentially generate entire assignment space
  $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_n$

$$\begin{aligned}\mathcal{D} &= \mathcal{D}_A \times \mathcal{D}_B \times \mathcal{D}_C \times \mathcal{D}_D \times \mathcal{D}_E \\ &= \{1,2,3,4\} \times \{1,2,3,4\} \times \ldots \times \{1,2,3,4\} \\ &= \left\{ \begin{array}{c} \langle 1,1,1,1,1\rangle \\ \langle 1,1,1,1,2\rangle \\ \ldots \\ \langle 4,4,4,4,4\rangle \end{array} \right\}\end{aligned}$$

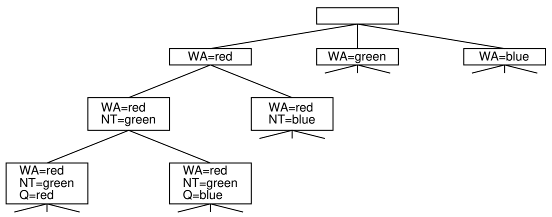
- Test each assignment against constraints.
- Generate-and-test is always exponential.
- . . . but see iterative repair techniques. . .

"Grow" Approach

Initial State: all variables are unassigned

Operators: assign value to any unassigned variable

Goal test: all variables are assigned and all constraints are satisfied



Complexity of the dumb approach

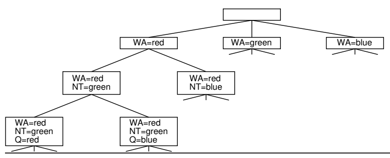
Max. depth of space $m =\ n$ (# of variables)

Depth of solution $d =\ n$ (all vars assigned)

Search algorithm to use: depth-first

Branching factor $b =\ \sum_i |D_i|$ (at top of tree)

Tricks for Grow



Improve dramatically by noting. . .

- Order of assignment is *irrelevant*
  so many paths are equivalent

  $\Rightarrow$ just consider assignment to variable X_i
  at depth i

- Adding assignments cannot correct
  a violated constraint

  $\Rightarrow$ stop&back-up whenever conflict

Other Tricks — Grow

1. Formulate CSProblem appropriately
   – Node = Variable, vs Node = Constraint
2. Prune domain
   – k-consistency
3. Avoid "Inconsistent" Values
   – Backtracking
   – Forward Checking
4. Best Variable/Value
   – Most-constrained variable first
   – Most-constraining variable first
   – Least-constraining value first

Trick#1a: Appropriate Formulation

- Place 8 queens (on 8×8 board)
  s.t. no queens attach another
  ∃! queen per row
  ∃! queen per column
  ∃! queen per ↗ diagonal
  ∃! queen per ↖ diagonal
- Naive approach: Use 8 variables

$$Q_i \quad \in \quad \left\{ \begin{array}{cccc} \langle 1,1\rangle & \langle 1,2\rangle & \ldots & \langle 1,8\rangle \\ \langle 2,1\rangle & \langle 2,2\rangle & \ldots & \langle 2,8\rangle \\ \vdots & \vdots & \vdots & \vdots \\ \langle 8,1\rangle & \langle 8,2\rangle & \ldots & \langle 8,8\rangle \end{array} \right\}$$

  Constraints:
  $Q_i \neq Q_j$, $Q_i.X \neq Q_j.X$, $Q_i.Y \neq Q_j.Y$, ...

But... must consider (discard) many silly assgn's
$Q_1 = \langle 1,1\rangle$ and $Q_2 = \langle 1,1\rangle$
$Q_1 = \langle 1,1\rangle$ and $Q_2 = \langle 1,2\rangle$
$Q_1 = \langle 1,1\rangle$ and $Q_2 = \langle 4,1\rangle$

- Better to AVOID "silly" assgn's
  ⋆ Only Generate 1 Queen per row
  ⋆ Only Generate 1 Queen per column

Reformulation of 8-Queens

- Use 8 variables $V_1, \ldots, V_8$,
  where $V_i \in \{1, \ldots, 8\}$ specifies COLUMN
  for queen on row i (Q_i)

  (get "∃! queen per <u>row</u>" for free)

- For "∃! queen per <u>column</u>":
  Force $\mathcal{V}$ to be PERMUTATION of $\{1, \ldots 8\}$

  (So exactly one V_i is 1, exactly one is 2, ...)

So: `solution( V ) :- permut([1,2,3,4,5,6,7,8], V), safe(V).`

`safe(V)` if no diagonal attacks.

8-Queens — Diagonal

- Q_i (queen on row i) attacks Q_j
  if
  $|V_i - V_j| \;=\; |i - j|$
- `V = [V1, ... V8]` is *SAFE* if
  Q_1 is NOT diag-attacking Q_2, Q_3, ...
  Q_2 is NOT diag-attacking Q_3, Q_4, ...
  ...

  Q_1 is NOT diag-attacking Q_2:
  `V1 - V2` $\neq$ 1, -1
  (as 1 row away, must not be 1 column away)

  Q_1 is NOT diag-attacking Q_3:
  `V1 - V3` $\neq$ 2, -2
  (as 2 rows away, must not be 2 columns away)

So: `safe(` [V_i | [V_{i+1}, ... , V_8]])
if
$V_i - V_{i+1} \neq \pm 1$
$V_i - V_{i+2} \neq \pm 2$
$\ldots V_i - V_{i+k} \neq \pm k$
... & `safe(` [V_{i+1}, ... , V_8])

Prolog Encoding — 8-Queens

```
solution( V ) :- permut([1,2,3,4,5,6,7,8], V), safe(V).

 %%%%%%%%%% Define Permutation:
permut([], []).
permut([A|Rest], Perm) :- permut(Rest, Prest), insert(A, Prest, Perm).

insert(A, L, [A|L]).
insert(A, [B|L1], [B|L2]) :- insert(A, L1, L2).

 %%%%%%%%%% Define Safe:
safe( [V] ).
      %% when only 1 queen remains, DONE!

 %% Safe if
  % 1. 1st Queen in list is NOT diag-attacking any in rest of list
  % 2. Rest of list is safe (recursively)
safe( [V | VList] ) :-
   nodiag(V, VList, 1),
   safe(VList).

  % if rest of list is NIL, can't be any attacks
nodiag( V, [], N).
  % ... else: see if queen on adjacent row is 1 column away,
  % then if queen 2 rows aways is 2 columns away, etc.
nodiag( V1, [V2 | List], N) :-
   noattack( V1, V2, N),
   N1 is N+1,
   nodiag(V1, List, N1).

noattack(V1, V2, N) :-
   N =\= V2-V1,
   N =\= V1-V2.
```


Comments on 8-Queens

- Still very inefficient,
  as considers FULL permutations,
  ...even if quickly pruned

Eg: Permutation [1,2,...] cannot work.
But alg will try [1,2,3,4,...],
then [1,2,3,5,...],
[1,2,3,6,...],
[1,2,3,7,...],
[1,2,3,8,...],
[1,2,4,3,...],
[1,2,4,5,...],
...

Better: When selecting position for k^{th} queen,
just make different from $V_1, \dots V_{k-1}$

- Soon see other ideas: ForwardChecking, ...

- n-queens: need generator ...
  `soln(N, S) :- gen(N, NTo1), permut(NTo1, S), safe(S).`

Trick#1b: Appropriate Formulation

- Crossword Puzzle:
  1. Var = Word (in Row/Column)
     Constraint = single $\langle i, j\rangle$ entry
     (eg, "3Down" and "5across" must have same $\langle 3, 5\rangle$ letter: $C_{3D,5A} = \{\langle \dots, \dots\rangle, \dots\}$)
     Only BINARY constraints
  2. Var = Letter at $\langle i, j\rangle$
     Constraint = consecutive letters in same word
     (eg, $L_{3,1}$, $L_{3,2}$, $L_{3,3}$ all form same word — $C_{31,32,33} \in \{\langle d, o, g\rangle, \langle c, a, t\rangle, \dots\}$)
     k-ary constraint, for k-letter word

- Exploit functions, ...

Trick#1b, con't

- n-ary vs 2-ary constraints
  Can always transform any n-ary csp to 2-ary
  Typically requires adding new variables...

- $C_{\alpha\beta\gamma} \quad \equiv \quad \alpha \oplus \beta = \gamma$

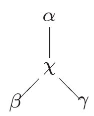
α	β	γ
0	0	1
0	1	0
1	0	0
1	1	1

- New variable: χ

χ	α	β	γ
A	0	0	1
B	0	1	0
C	1	0	0
D	1	1	1

$$\begin{aligned} C_{\chi\alpha} &= \{\langle A,0\rangle, \ \langle B,0\rangle, \ \langle C,1\rangle, \ \langle D,1\rangle\ \} \\ C_{\chi\beta} &= \{\langle A,0\rangle, \ \langle B,1\rangle, \ \langle C,0\rangle, \ \langle D,1\rangle\ \} \\ C_{\chi\gamma} &= \{\langle A,1\rangle, \ \langle B,0\rangle, \ \langle C,0\rangle, \ \langle D,1\rangle\ \} \end{aligned}$$

- Variable $\leftrightarrow$ Constraint



For Trick#2: Scheduling Activities

- **Variables**: A, B, C, D, E
  (starting time of activity)

- **Domains**: $\mathcal{D}_i = \{1,2,3,4\}$, for $i = A, B, \ldots, E$

- **Constraints**:

  $(B \neq 3)$, $(C \neq 2)$, $(A \neq B)$, $(B \neq C)$,
  $(C < D)$, $(A = D)$, $(E < A)$, $(E < B)$,
  $(E < C)$, $(E < D)$, $(B \neq D)$.

Prolog Encoding – Schedule

```
  %%% Domain information
domA(A) :- v14(A).
domB(B) :- v14(B).
domC(C) :- v14(C).
domD(D) :- v14(D).
domE(E) :- v14(E).

 % v14(X) :- member(X, [1,2,3,4] ).
v14(1).   v14(2).   v14(3).  v14(4).

  %%% Constraint information
cB(B) :- B \== 3.
cC(C) :- C \== 2.
cAB(A,B) :- A \== B.
cBC(B,C) :- B \== C.
cCD(C,D) :- C < D.
cAD(A,D) :- A = D.
cAE(A,E) :- E < A.
cBE(B,E) :- E < B.
cCE(C,E) :- E < C.
cDE(D,E) :- E < D.
cBD(B,D) :- B \== D.

  %%% Total constraint:
ans([A,B,C,D,E]) :-
    domA(A), domB(B),  domC(C),
    domD(D), domE(E),
    cB(B),    cC(C),    cAB(A,B), cBC(B,C),
    cCD(C,D), cAD(A,D), cAE(A,E), cBE(B,E),
    cCE(C,E), cDE(D,E), cBD(B,D).

 % bagof( X, ans(X), S).
 % S = [[4,2,3,4,1]] ? ;
```


Trick#2: Prune domain

2. **Consistency**: Prune variable's domain, before selecting value.

- *Arc-consistency:*

  Given *binary-constraint* $C_{X,Y}$
  $\mathcal{D}_X, \mathcal{D}_Y$ are **arc consistent** (or **2-consistent**)
  if
  $\forall x \in \mathcal{D}_X \quad \exists y \in \mathcal{D}_Y$ s.t. $\langle x, y \rangle \in C_{X,Y}$

  Eg: $\mathcal{D}_A = \{1,2,3,4\}$ and $\mathcal{D}_E = \{1,2,3,4\}$ are
  NOT arc consistent as $A = 1$ is not consistent with $E < A$

$\Rightarrow$ use $\mathcal{D}'_A = \{\ 2,3,4\}$ and $\mathcal{D}'_E = \{1,2,3\ \}$

Trick#2: con't

2. **Consistency**: Prune variable's domain, before selecting value.

- Given *unary-constraint* C_X
  $\mathcal{D}_X$ is **domain consistent** (or **1-consistent**)
  if
  $\forall x \in \mathcal{D}_X,\ x \in C_X$

  Eg: $\mathcal{D}_B = \{1,2,3,4\}$ is NOT domain consistent
  as $B = 3$ violates constraint $B \neq 3$.

  $\Rightarrow$ use $\mathcal{D}'_B = \{1,2,4\}$

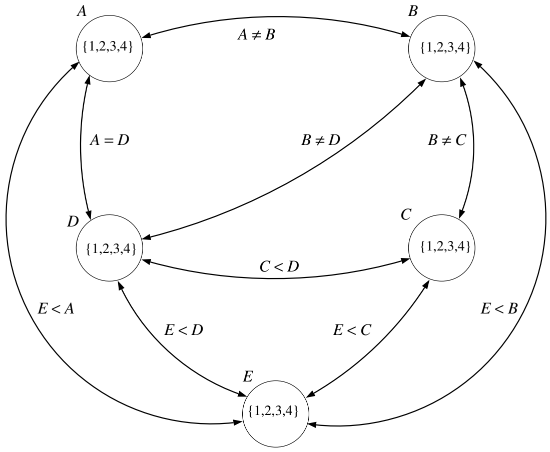
- Given *binary-constraint* $C_{X,Y}$
  $\mathcal{D}_X, \mathcal{D}_Y$ are **arc consistent** (or **2-consistent**)
  if
  $\forall x \in \mathcal{D}_X \quad \exists y \in \mathcal{D}_Y$ s.t. $\langle x, y \rangle \in C_{X,Y}$

  Eg: $\mathcal{D}_A = \{1,2,3,4\}$ and $\mathcal{D}_E = \{1,2,3,4\}$ are
  NOT arc consistent as $A = 1$ is
  not consistent with $E < A$

  $\Rightarrow$ use $\mathcal{D}'_A = \{2,3,4\}$ and $\mathcal{D}'_E = \{1,2,3\}$

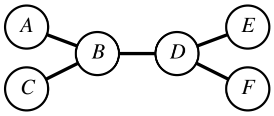
- Given *k-ary–constraint* $C_{X_1,\ldots,X_k}$
  $\mathcal{D}_{X_1}, \ldots, \mathcal{D}_{X_k}$ are k **consistent** if
  $\forall x_1 \in \mathcal{D}_{X_1} \quad \exists x_2, \ldots, x_k \in \mathcal{D}_{X_2}, \mathcal{D}_{X_3}, \ldots, \mathcal{D}_{X_k}$ s.t.
  $\langle x_1, x_2, \ldots, x_k \rangle \in C_{X_1,\ldots,X_k}$

Binary Constraints ⇒ Constraint Network



Note: Already removed $B = 3$, $C = 2$.

Special Case: Tree structured CSPs



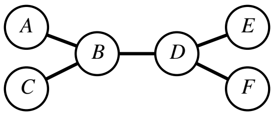
- Recall for general CSPs:
  worst-case time is $O(|D|^n)$

Theorem: If the constraint graph is tree-structured
(has no loops),
Arc-Consistency is sufficient!
⇒ CSP can be solved in $O(n|D|^2)$ time.

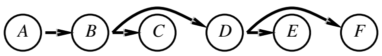
- Also applies to
  logical / probabilistic reasoning

- Important example of relation between
  syntactic restrictions and
  complexity of reasoning

Algorithm for Tree-Shaped CSP



1. Order nodes breadth-first, starting from any leaf:



2. For $j = n$ to 1, apply $AC(V_i, V_j)$
   where V_i is parent of V_j

3. For $j = 1$ to n, pick legal value for V_j, given parent value

Trick#3: Avoid "Inconsistent" Values

- **Backtracking**: Only consider $X_i = v$ if consistent w/ *earlier* assignments
  $\langle X_1 = v_1, \ldots, X_{i-1} = v_{i-1} \rangle$

  Eg: Given $\langle A = 1 \rangle$, do NOT allow $B = 1$, as violates $A \neq B$.

- **Forward Checking**: If considering $X_i = v$, remove from $\mathcal{D}_j$ $(j > i)$ any no-longer-possible value
  . . . make arc-consistent . . .

  If $\exists j$ s.t. $\mathcal{D}_j \mapsto \{\}$, disallow $X_i = v$.

  Eg: Spse $\mathcal{D}_D = \{2, 3, 4\}$ and $C_{A,D} \equiv$ "$A = D$"
  – Do NOT consider $A = 1$, as violates $A = D$.
  – After $A = 2$, change $\mathcal{D}_D = \{2\}$

Note: AC is "preprocessing step",
but FC is done "during the computation"

AC will keep propagating (until CSP is arc-consistent)
but FC is just current-var with future vars

(. . . but FC could continue propagating. . .)

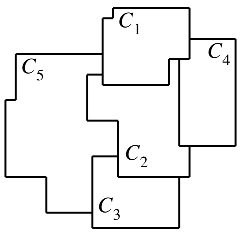
Illustrating ForwardChecking #1

Idea: Keep track of remaining legal values
for each unassigned variable

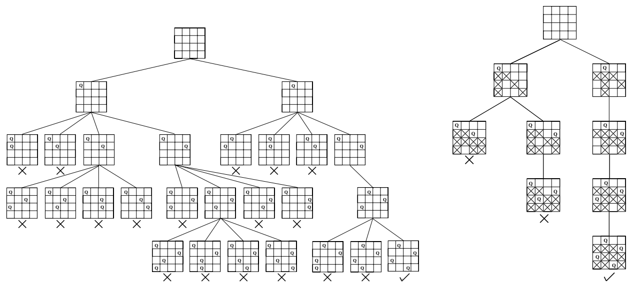
Terminate search when
any variable has no legal values

- Simplified map-coloring example:

	red	blue	green
C_1			
C_2			
C_3			
C_4			
C_5			



Illustrating ForwardChecking #2



Can be EXPONENTIAL win:

Given CSP on $\{X_1, X_2, \ldots, X_n\}$

Spse $\{X_1, X_2, X_n\}$ all $\{1, 2\}$,

$C_{1,2,n} \equiv$ "$x_1 \neq x_2,\ x_1 \neq x_n,\ x_2 \neq x_n$"

Trick#4: Best Variable/Value

- Earlier: removing impossible situations.
  Other decisions:

4. **Most-constrained variable first:**
Select unassigned variable with smallest domain
(dynamic: after each pruning w/forward checking, ...)

   Eg: If $|\mathcal{D}_E| = 2$ and $|\mathcal{D}_i| \geq 3$ for other i, select E

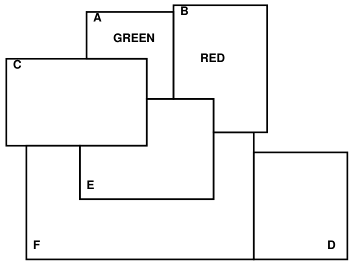
5. **Most-constraining variable first:**
Select unassigned variable that appears in most constraints w/ other unassigned variables

   Let $f(X) = |\{Y \ : \ Y \ \text{ unassigned}, \exists C_{\ldots X \ldots Y}\}|$
   Select $X^* = \arg\min_X\{\ f(X) \ : \ X \text{unassigned} \ \}$

   Eg: Start with B, as $f(B) = 4 \geq f(X)\ \forall X$, ...

6. **Least-constraining value first**:
Choose value for X that *leaves* the most values
for OTHER unassigned variables

Example of Heuristics



Most Constrained Variable:
E has only 1 legal value (Blue);
C has 2, F has 2, D has 3
⇒ pick E

Most Constraining Variable:
C "is constrained by" E, F
E "is constrained by" C, F
F "is constrained by" C, F, D
D "is constrained by" F
⇒ pick F

Least Constraining Value:
Q: What color for C?
Both `Blue` and `Red` are legal.
A: `Red` leaves most option for E...

How Effective are Heuristics?

Consider n-queens:

- with ForwardChecking: $n = 30$

\+ Most-Constrained-Variable: $n = 100$

\+ Least-Constrained-Value: $n = 1000$

- Dramatic recent progress in
  Constraint Satisfaction

- ... can now handle problems
  - with **10,000** to **100,000** variables
  - up to **1,000,000** constraints!

(see p.105 R&N)

Hard CSPs

- Suppose all constraints UNARY
  (explicit)

⇒ Trivial to solve
1,000,000,000 variable system
w/ **10,000,000,000** (such) constraints!

But...

- Job-Shop Scheduling:
  10 jobs on 10 machines

- Proposed [Fisher/Tompson: **1963**]

- Solved [Carlier/Pinson: **1990**]

- Open: 15 jobs on 15 machines

Constraint Optimization Problem

- So far... *SATISFACTION*
  What about *OPTIMIZATION*?

- Want to minimize $\left\{ \begin{array}{l} \#\text{ of rooms required} \\ \#\text{ chip size} \\ \#\text{ time for delivery} \end{array} \right.$

- Obvious approach:

```
Set try_time = t_max
Set best_time = "None"
Repeat
    Add constraint   Time < try_time
      to existing constraints
    Try to find satisfying solution.
    If satisfied,
      Set best_time = try_time
      Set try_time = try_time - 1
    Else Return( best_time )
```


Comments

- Why not Mathematical Programming Problem?
  - CSP rep'n more natural/expressive
    + variables ≈ problem entities
    + constraints ≈ natural description
      (not just linear inequalities)

    ⇒ Formulation simpler, solution easier to understand, easier to find good heuristics
  - CSP algorithms often find sol'n faster

- ∃ ConstraintProblemSolving tools/systems
  + CHIP ("Constraint Handling in Prolog")
  + PrologIII
  + Solver (from ILOG)

- Tools use general, "weak" methods
  If have background knowledge: use it!
  Symmetries

  Clearly T is even in...

```
   G E R A L D
 + D O N A L D
 -------------
   R O B E R T
```

- Other tricks (backjumping, dynamic ...)
  + theoretical analyses

- Auxilary paper on course web-page

How to use PROLOG FDS to solve Constraint Problems?

0. Load "Sicstus Prolog's Finite Domain Solver"
```
:- use_module(library(clpfd)).
```

1. Declare domains
```
domain([X,Y], 1,4)
   variables X, Y have domain {1,2,3,4}
domain([Z], 3,5)
   variable Z has domain {3,4,5}
   domain values = consecutive integers
```

2. specify constraints
```
A #> 3
A #< B
N #= C0 + C1*N1 + C2*N2*N2
noLabeling(N,L) :-   L = [A,B,C],
   domain([A,B,C],1,N),  A #> B, B #> C.
```

3. Prolog assigns domain values to variables, 1-by-1, until finding satisfying ass't
```
| ?- noLabeling(4,L).
L = [_A,_B,_C],
_A in 3..4,
_B in 2..3,
_C in 1..2 ?
```


Word Problem: Which Bottle?

To leave a maze, need to select/drink bottle one of 7 bottles, in left to right line...

1 Danger lies before you, while safety lies behind,
2 Two of us will help you, whichever you would find,
3 One among us seven will let you move ahead,
4 Another will transport the drinker back instead,
5 Two among our number hold only nettle wine,
6 Three of us are killers, waiting hidden in line.
7 Choose, unless you wish to stay here forevermore,
8 To help you in your choice, we give your these clues four:
9 First, however slyly the poison tries to hide
10 You will always find some on nettle wines left side;
11 Second, different are those who stand at either end,
12 But if you would move onwards, neither is your friend;
13 Third, as you see clearly, all are different size,
14 Neither dwarf nor giant holds death in their insides;
15 Fourth, the second left and the second on the right
16 Are twins once you taste them, though different at first sight.

Prolog Encoding#2: SEND+MORE=MONEY

```
sum(N1, N2, N) :- sum1(N1, N2, N,
    0, 0,      % cariers from right, and to left
    [0,1,2,3,4,5,6,7,8,9], _ ).

sum1( [], [], [], 0, 0, Digits, Digits).

 % sequentially find values of digits...
sum1( [D1|N1], [D2|N2], [D|N], C1, C, Dig1, Digs) :-
   sum1( N1, N2, N, C1, C2, Dig1, Digs2),
   digitsum( D1, D2, C2, D, C, Digs2, Digs).

digitsum( D1, D2, C1, D, C, Digs1, Digs) :-
  del(D1, Digs1, Digs2),
  del(D2, Digs2, Digs3),
  del(D , Digs3, Digs),
  S = D1 + D2 + C1,
  D is S mod 10,
  C is S // 10.

%% If ''A'' is assigned, do nothing.
%% Else, set A to an element in L,

del(A, L, L) :- nonvar(A).  % A already instantiated

del( A, [A|L], L).    % delete the head

del( A, [B|L], [B|L1]) :- del(A, L, L1).
```