RN, Chapter 5

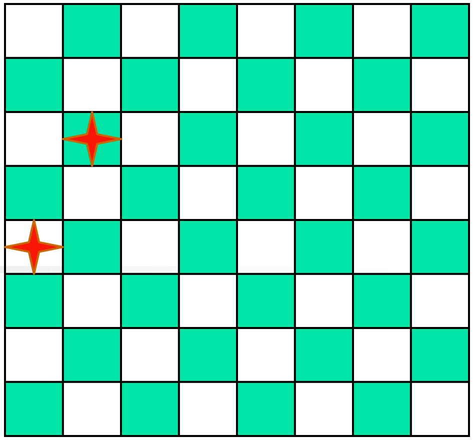
Constraint Satisfaction Problems

Some material from: D Lin, J You, JC Latombe

Search Overview

- Introduction to Search
- Blind Search Techniques
- Heuristic Search Techniques
- Constraint Satisfaction Problems
  - Motivation, Examples, Def’n
  - Complexity
  - Solving:
    - Formulation, Propagation, Heuristics
  - Special Case (tree structured)
  - Constraint Optimization Problems
  - Example: Edge labeling
- Local Search Algorithms
- Game Playing search

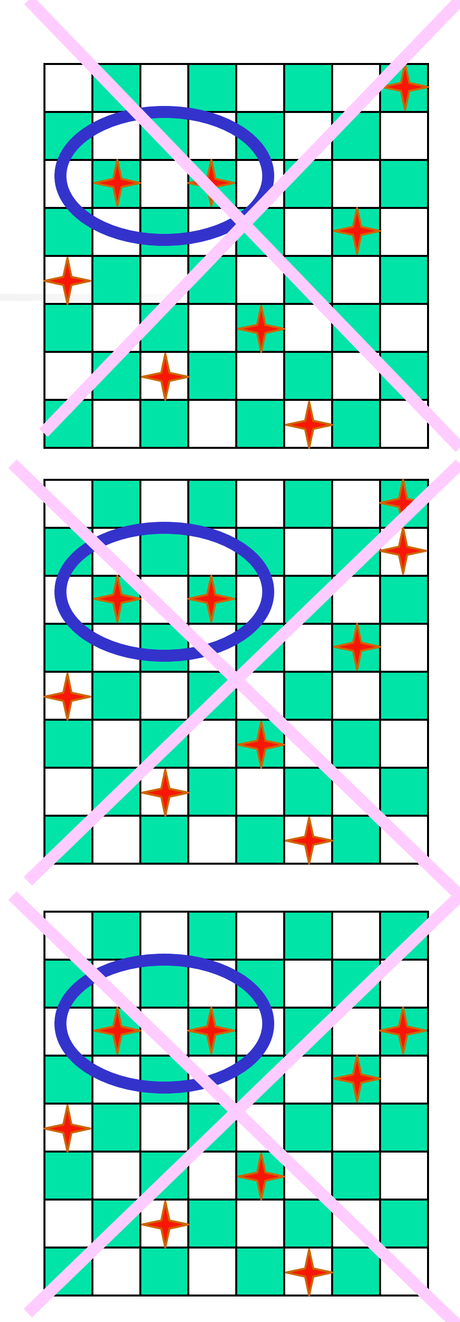
Example: 8-Queens



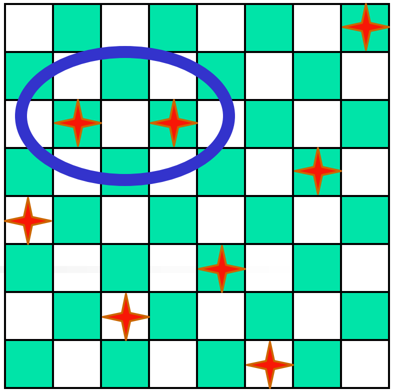
- Place 8 Queens on board s.t.
  No two Queens attack each other
- ≡ Constraint Satisfaction Problem
  - Find assignment $\{ Q_i := r_i \}$ satisfying
  - Set of given constraints
    - Q_3 and Q_7 cannot both be on column 4
    - Q_3 and Q_8 cannot both be on column 5
    - …

Naïve Algorithm

- Initialize the queens: $\forall i \;\; Q_i := 1$
- While assignment is not ok:
  - Increment $Q_8 := Q_8+1$
  - If $Q_8=9$:
    - $Q_8 := 1$; $Q_7 := Q_7 +1$
    - If $Q_7=9$:
      - $Q_7 :=1$; $Q_6 := Q_6 +1$
      - If …
- Return assignment



Problem with Naïve Algorithm



- Consider assignment

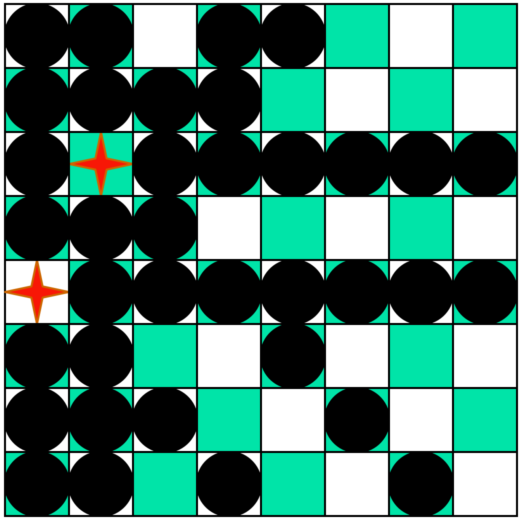
  $\{ Q_1=5, Q_2=3, Q_3=7, Q_4=3, \ldots\}$

  - Note queens Q_2 and Q_4 attack one another
- … sufficient to declare this entire assignment BAD!
- So can make LOCAL decisions:
  - Assign queens SEQUENTIALLY
  - Stop as soon as find ANY violation

Better Approach for 8-Queens

Assign queens sequentially
 (from left to right)

Only assign queens to
 LEGAL positions



What is Needed?

- States, Actions, Goal test…
- Also:
  - an early failure test
    (based on partial assignment)
  - a way to propagate the constraints imposed by one queen on the others
    … using partial assignment to constrain remaining assignments …

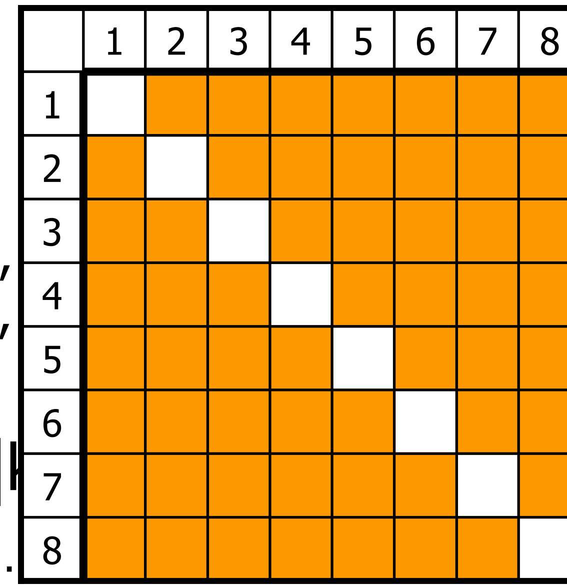
→ Explicit representation of constraints and constraint manipulation algorithms

Constraint Satisfaction Problem

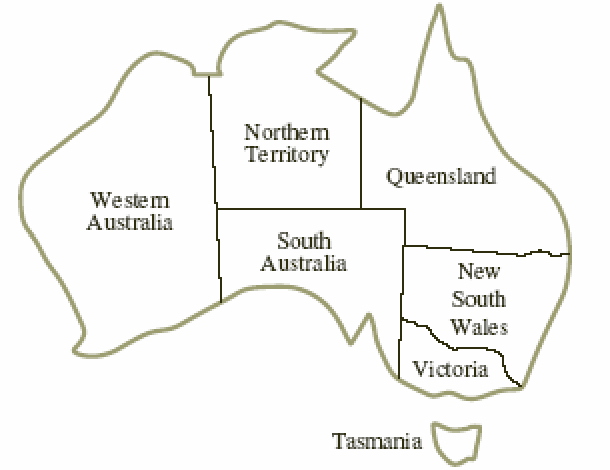
- Set of variables $\{X_1, X_2, \ldots, X_n\}$
  - Each X_i has domain D_i of possible values
  - (Here: D_i is discrete, finite)
- Set of constraints $\{C_1, C_2, \ldots, C_p\}$
  Each C_k …
  - specifies allowable combinations of values of…
  - a subset of variables
- SOLN: Assign a value to every variable, such that *all* constraints are satisfied

Example: 8-Queens Problem

- 8 variables X_i, i = 1 to 8
- Domain for each variable: {1,2,…,8}
- Constraints of the forms:
  - $\forall i, j \neq i, k \quad X_i = k \rightarrow X_j \neq k$
    - C_{12} : $(X_1, X_2) \in$ { (1,2), (1,3), …, (1,8), (2,1), (2,3), …, (2,8), (8,1), …, (8,7)
  - $\forall i, j \neq i, k_i, k_j \quad X_i = k_i, X_j = k_j \rightarrow |i-j| \neq |k$
    - C'_{13}: $(X_1, X_3) \in$ { (1,1), (1,2), (1,4), … (2,1), (2,2), (2,3), (2,5), …, (2,8), (3,2), (3,3), (3,4), - (3,8), … (8,8) }



Example: Map Coloring



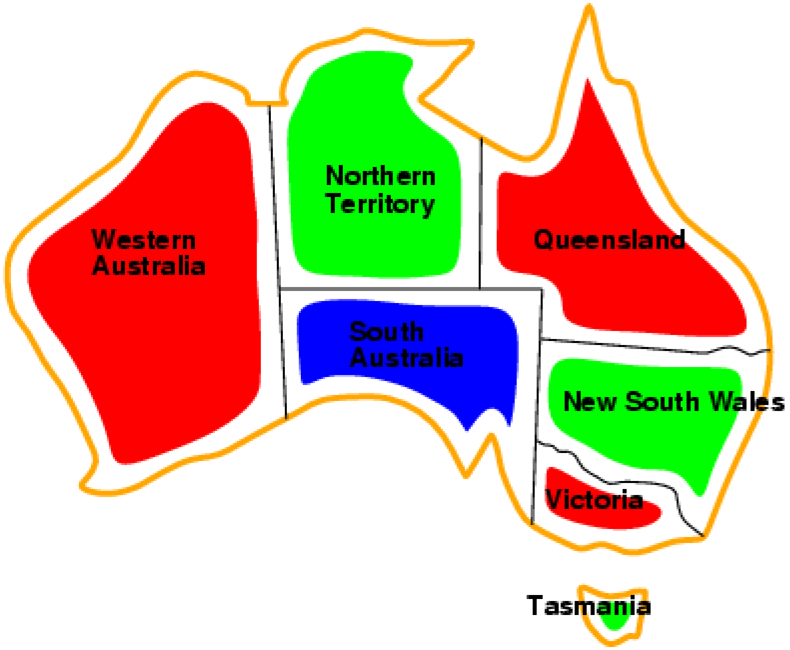
- Color "map" s.t.
  Adjacent "regions" have different colors
- ≡ Constraint Satisfaction Problem
  - Find assignment (color to each region) satisfying…
  - Set of given constraints
    - Region WA cannot be same color as SA
    - Region WA cannot be same color as NT
    - …

$WA \neq NT$, $WA \neq SA$, $NT \neq SA$ $NT \neq Q$, $SA \neq Q$, $SA \neq NSW$, $SA \neq V$, $Q \neq NSW$, $NSW \neq V$

Example: Map Coloring

- 7 **Variables**: { V, T, WA, NT, SA, Q, NSW }
- **Domains**: Di = { r, g, b }
(same domain for each)
- **Constraints**: Adjacent regions must have different colors
- $C_{WA,NT}$ constrains values for WA and NT:
$C_{WA,NT}$: (WA,NT) $\in$ { [r,g], [r,b], [g,r], [g,b], [b,r], [b,g] }
- Similarly:
$C_{WA,NT}$, $C_{WA, SA}$, $C_{NT, SA}$, $C_{NT,Q}$, $C_{SA,Q}$, $C_{SA,NSW}$, $C_{SA,V}$, $C_{Q,NSW}$, $C_{NSW,V}$

Example: Map-Coloring



- Solutions ≡ complete and consistent assignment
  - e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Puzzles…

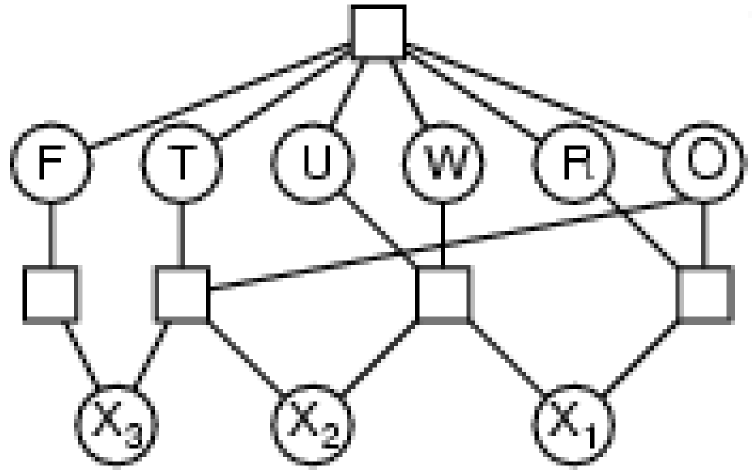
To leave a maze, need to select/drink bottle one of 7 bottles, in left to right line. . .

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

***Harry Potter and Sorcerer's Stone*, JK Rowling**

Example: Cryptarithmetic

```
   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: *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

Cryptoarithmetic

$$\begin{array}{r} \text{SEND} \\ +\ \text{MORE} \\ \hline \text{MONEY} \end{array}$$

- Map LETTER to DIGITS 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 Version#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

  $S \neq E,\ S \neq N, \ldots$
- **Constraints Version#2**:

  $D + \ = Y + 10 \times c_1$

  $N + R + c1 = E + 10 \times c_2$ (c_i is "carry")

  $E + O + c2 = N + 10 \times c_3$

  $S + M + c3 = O + 10 \times\ M$

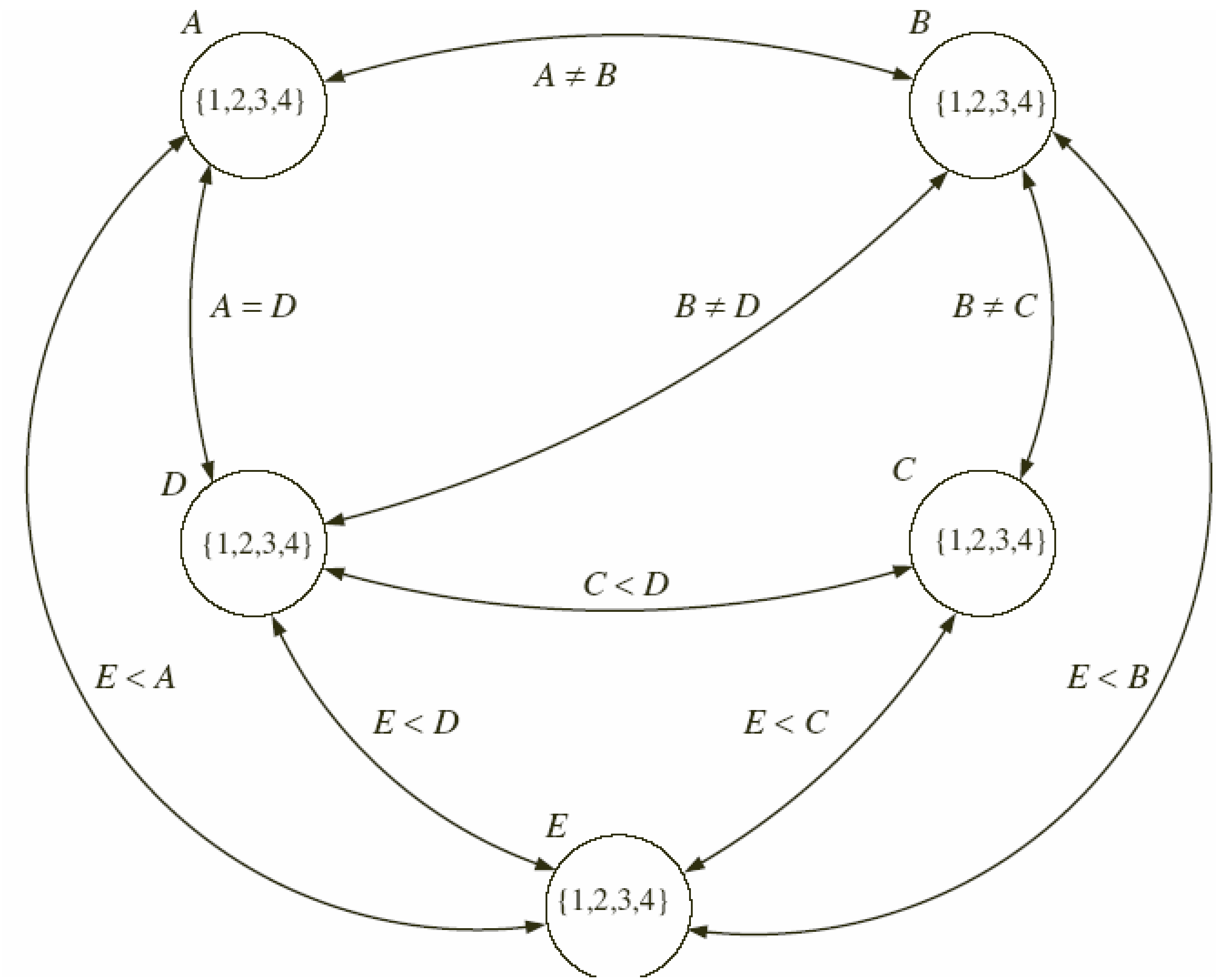
+ Unique: each letter is different . . .

Example: Scheduling Activities

- Variables: *A, B, C, D, E*
  (starting time of activity)
- Domains: $D_i = \{1, 2, 3, 4\}$,
  for i = *A, B, ..., E*
- Constraints:
  $(B \neq 3), (C \neq 2), (A \neq B), (B \neq C),$
  $(C < D), (A = D), (E < A), (E < B),$
  $(E < C), (E < D), (B \neq D)$

"A = D" $\equiv$ { [1,1], [2,2], [3,3], [4,4] }
"E < A" $\equiv$ { [1,2], [1,3], [1,4], [2,3], [2,4], [3,4] }

Constraint Network

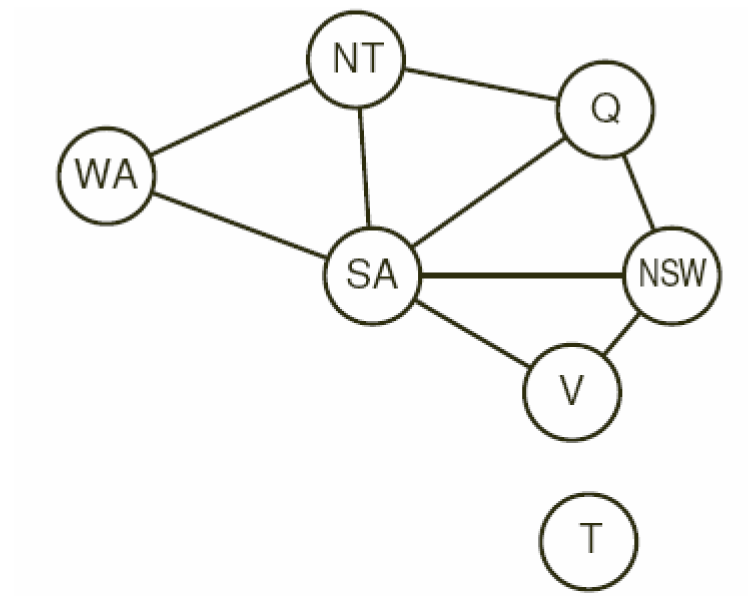


Examples

- Assignment problems
  - . . . who teaches what class
- Time-tabling problems
  - . . . which class is offered when & where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Map-coloring
- Crypto-arithmetic

Constraint GRAPH

- If constraints all BINARY
  - (relate 2 variables)
- Connect variables by an edge if in constraint



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

Complexity of CSP

- Propositional Satisfiability is CSProblem
  - Domain of each variable: {t, f}
  - Each k-clause allows 2^k -1 assignments, …)

$\Rightarrow$ Every NP-complete problem can be formulated as CSProblem.

. . . so CSPs are HARD to solve!

Approaches

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

Search Approaches to CSP

1. "Modify/Repair"
   - State: complete assignment
     Initial state: random(?)
   - Operator: Change value of some variable
2. "Grow"
   - State: partial assignment
     Initial state: ⟨⟩
   - Operators:

     1. Assign value to any unassigned variable
        Branching Factor: $\sum_i |D_i|$

     2. Assign value to k +1st variable
        (Branching Factor: $\max_i |D_i|$

- + … in all ca

  - Goal-test: a
  - PathCost: 0

Goal test is DECOMPOSED into individual constraints
If A ≠ B,
then ⟨ A = 1, …, B = 1, …⟩ cannot be part of solution…
⇒ 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
  $D = D_1 \times D_2 \times \ldots \times D_n$

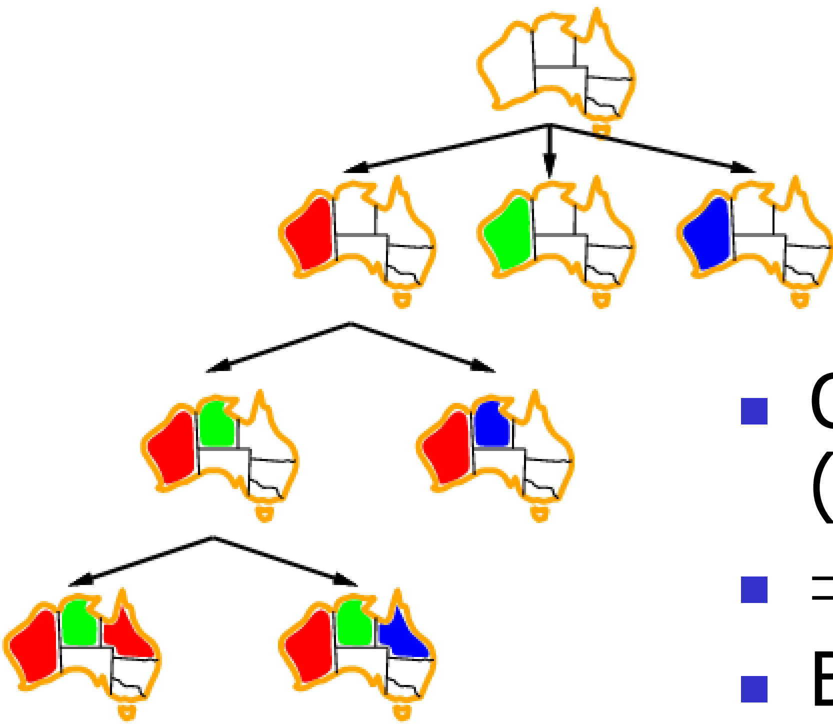
- Eg: $D = D_A \times D_B \times D_C \times D_D \times D_E$
  $= \{1,2,3,4\} \times \{1,2,3,4\} \times \ldots \times \{1,2,3,4\}$
- Test each assignment against constraints
- Generate-and-test is always exponential
- ... but see "Local Search Algorithms" ...

"Grow" Approach

- Initial state: empty assignment { }
- Successor function:
  - assign a value to an unassigned variable
  - … which does not conflict with the currently assigned variables
- Goal test: the assignment is complete
- Path cost: irrelevant
  - ie, "0"

Every solution involves *n* variables, appears at depth *n*
→ use depth-first search

CSP as Search



- Only depth-n search (n variables)
- $\Rightarrow$ So DFS is standard…
- Branching factor: $\max_i |D_i|$
  - Why not $\sum_i |D_i|$?

Backtracking Algorithm

CSP-BACKTRACKING(PartialAssignment a)

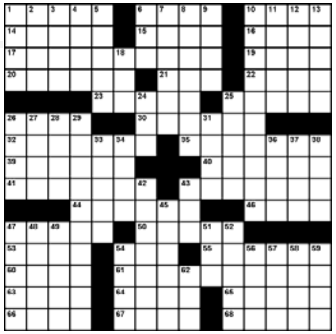
- If a is complete, then return a
- X ← select an unassigned variable
- D ← select an ordering for the domain of X
- For each value v in D do
  - If v is consistent with a then
    - result ← CSP-BACKTRACKING(a + (X= v))
    - If result ≠ *failure* then return result
- Return *failure*

CSP-BACKTRACKING({})

Improving “Grow” Approach

1. Formulate CSProblem appropriately
   - Node = Variable, vs Node = Constraint
2. Avoid “Inconsistent" Values
   - Backtracking
   - Forward Checking
3. Prune domain
   - Arc consistency
   - MAC
   - Interleave Assign/MAC
4. Heuristics: Best Variable/Value
   - Most-constrained variable first
   - Most-constraining variable first
   - Least-constraining value first

Trick#1: Appropriate Formulation



Crossword Puzzle:

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

Trick#1: con't: n-ary vs 2-ary constraints

- Can transform any n-ary csp to 2-ary
- Typically requires adding new variables…

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

α	β	γ
0	0	1
0	1	0
1	0	0
1	1	1

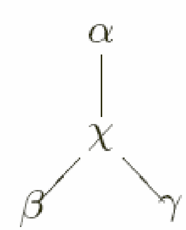
3-ary!

- New variable: χ

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

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

Each is binary



- Variable $\leftrightarrow$ Constraint

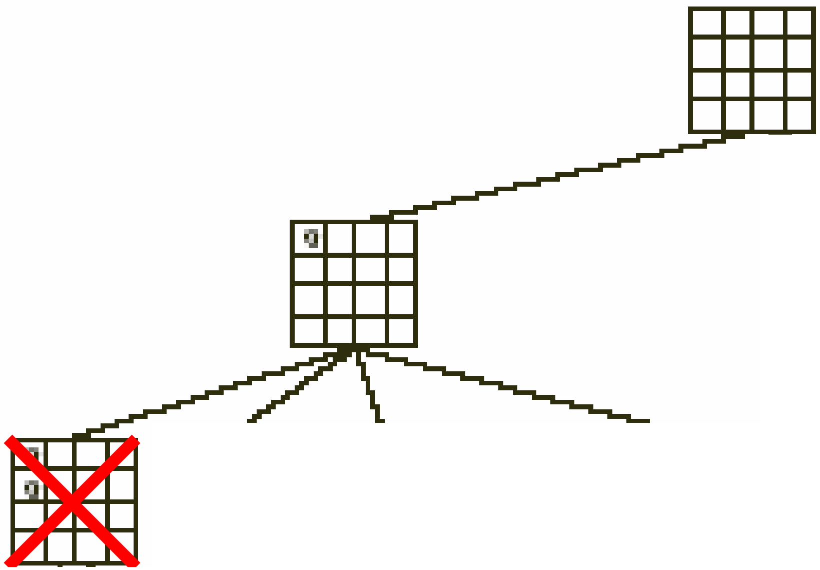
Cross-word puzzle: letter vs word…

Trick#2: Avoid "Inconsistent" Values

Backtracking

- If inconsistent, undo last assignment
- Reach X_i via path $\langle X_1=v_1, \ldots, X_{i-1} = v_{i-1}\rangle$
- If $X_i=v$ inconsistent,
  back up… try another $X_i=v'$
- If no value of X_i consistent,
  back up to reset earlier var
  - …try some OTHER value for $X_{i-1} = v'_{i-1}$
- Eg: Given constraints "$A\neq C$","$B>C$", $D_C=\{1,2,3,4\}$
  - After $\langle A=1, B=2 \rangle$, no legal values for C
  - ⇒ BACKTRACK to B… reset $B=3$ … $\langle A=1, \mathbf{B=3} \rangle$
  - ⇒ … now can use $C=2$

Backtracking



Trick#2: Avoid "Inconsistent" Values

Forward Checking:

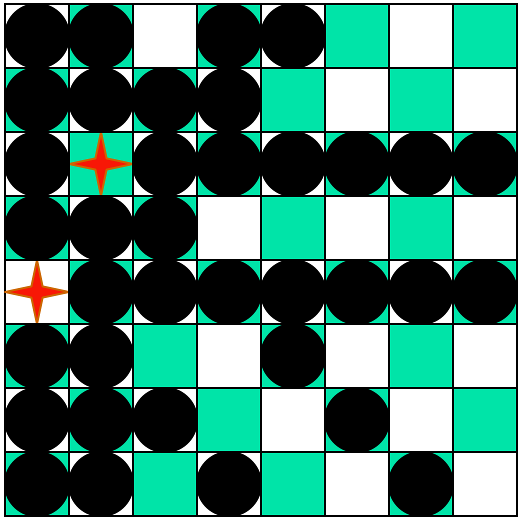
- After assign $X_i = v$,
- remove from D_j $(j > i)$
  any no-longer-possible value
  . . . make arc-consistent (wrt X_i). . .
- If $\exists$ j s.t. $D_j \mapsto \{\}$, disallow $X_i = v$

Eg: Spse $C_{A,D} \equiv$ "$A = D$"; $D_D = \{2,3,4\}$

  - Do NOT consider $A = 1$, as violates $A = D$
  - After $A = 2$, change $D_D := \{2\}$

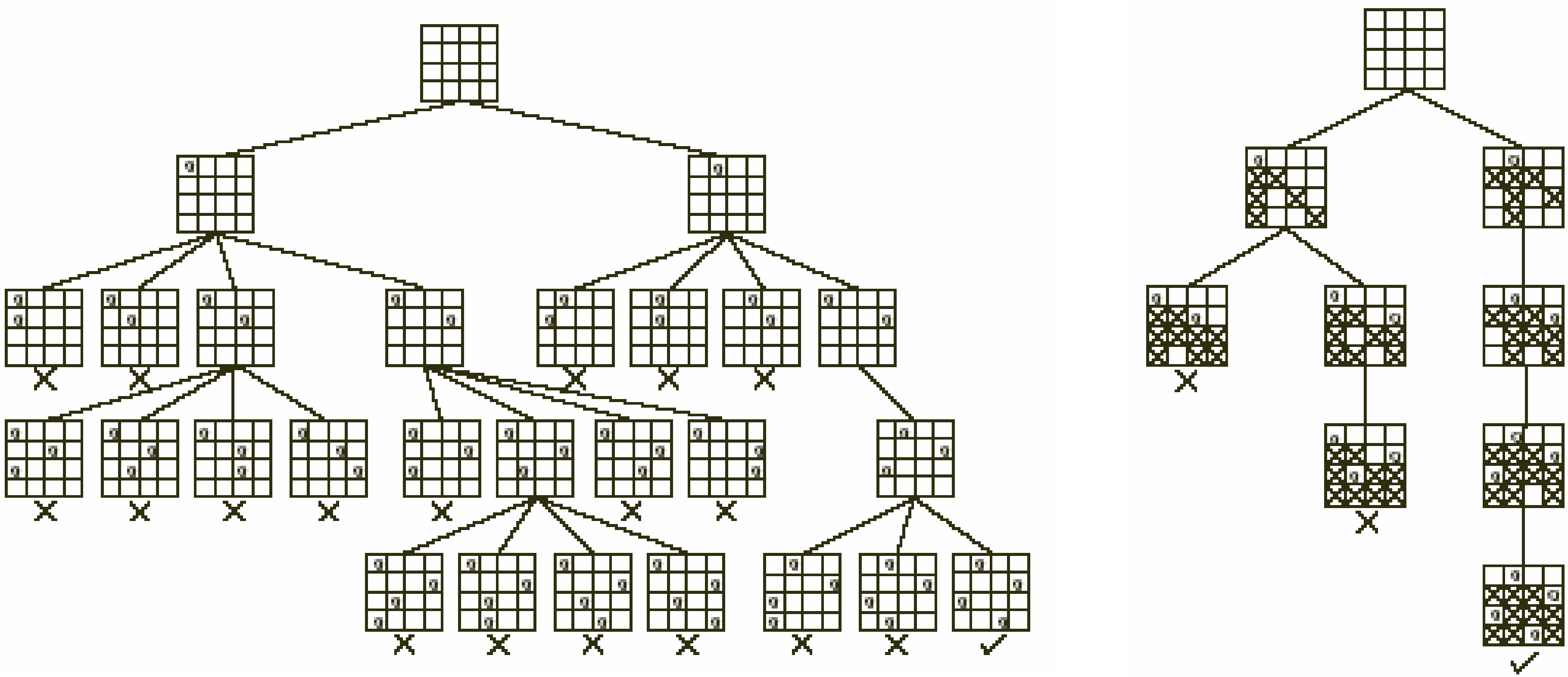
Illustrating ForwardChecking #1

If considering X := v,
consider each unassigned variable Y
that is connected to X by a constraint and
delete from Y's domain any value that is
inconsistent with v



Illustrating ForwardChecking #2

+ Forward Check

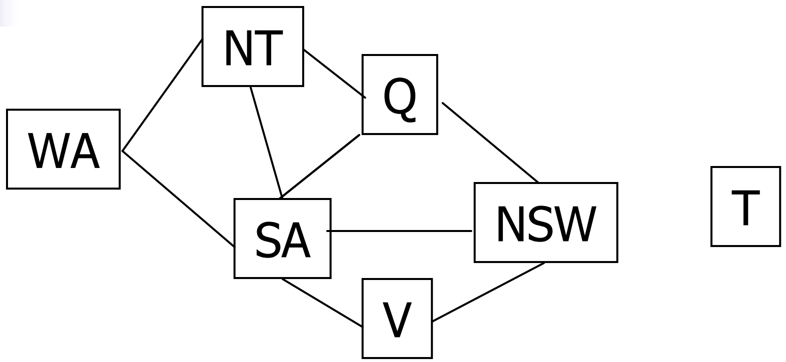


Illustrating ForwardChecking #3

Can be EXPONENTIAL win:

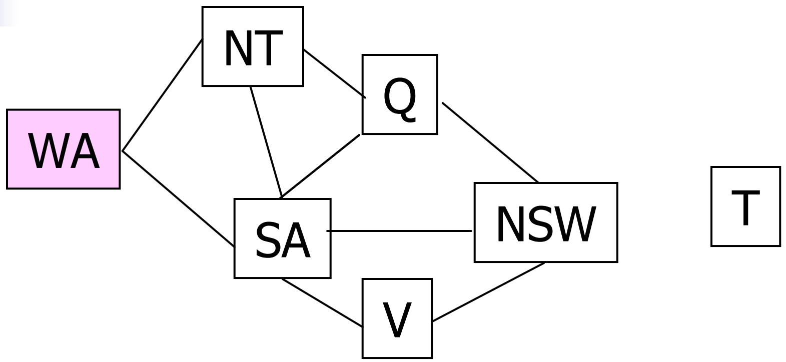
- CSP on $\{X_1, X_2, \ldots, X_n\}$
- Each of $\{X_1, X_2, X_n\}$ is $\{1, 2\}$
- $C_{1,2,n} \equiv$ "$x_1 \neq x_2 \,\&\, x_1 \neq x_n \,\&\, x_2 \neq x_n$"

Illustrating ForwardChecking #4



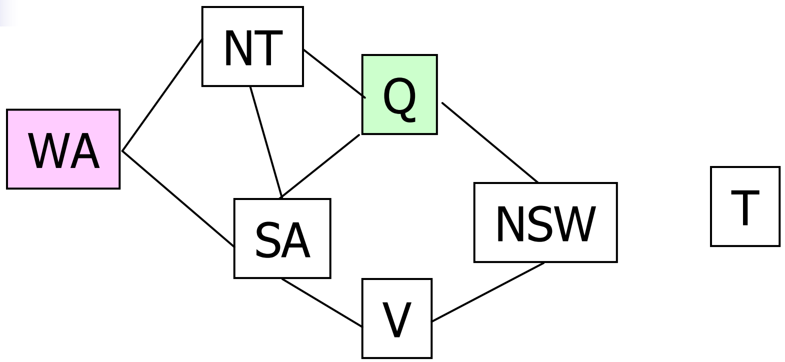
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

Illustrating ForwardChecking #4



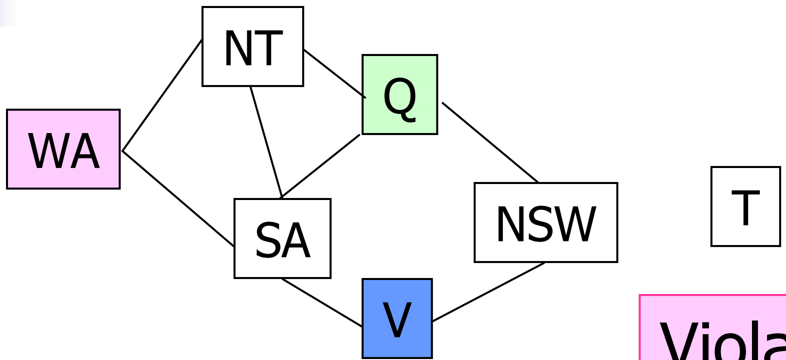
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB

Illustrating ForwardChecking #4



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	R B	RGB	B	RGB

Illustrating ForwardChecking #4



Violation that forward checking does not detect

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	R B	RGB	B	RGB
R	B	G	R	B		RGB

So… cannot set V=B here!

Forward Checking is not enough…

- FC propagates assignment to current-variable,
  to future variables
- Not sufficient!!
- Extensions:
  - “Preprocessing step“ – ArcConsistency
  - More elaborate propagation “during the computation"

Trick#3: Prune domain

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

- Arc-consistency:

  Given binary-constraint $C_{X,Y}$:

  D_X, D_Y are arc consistent (or 2-consistent)

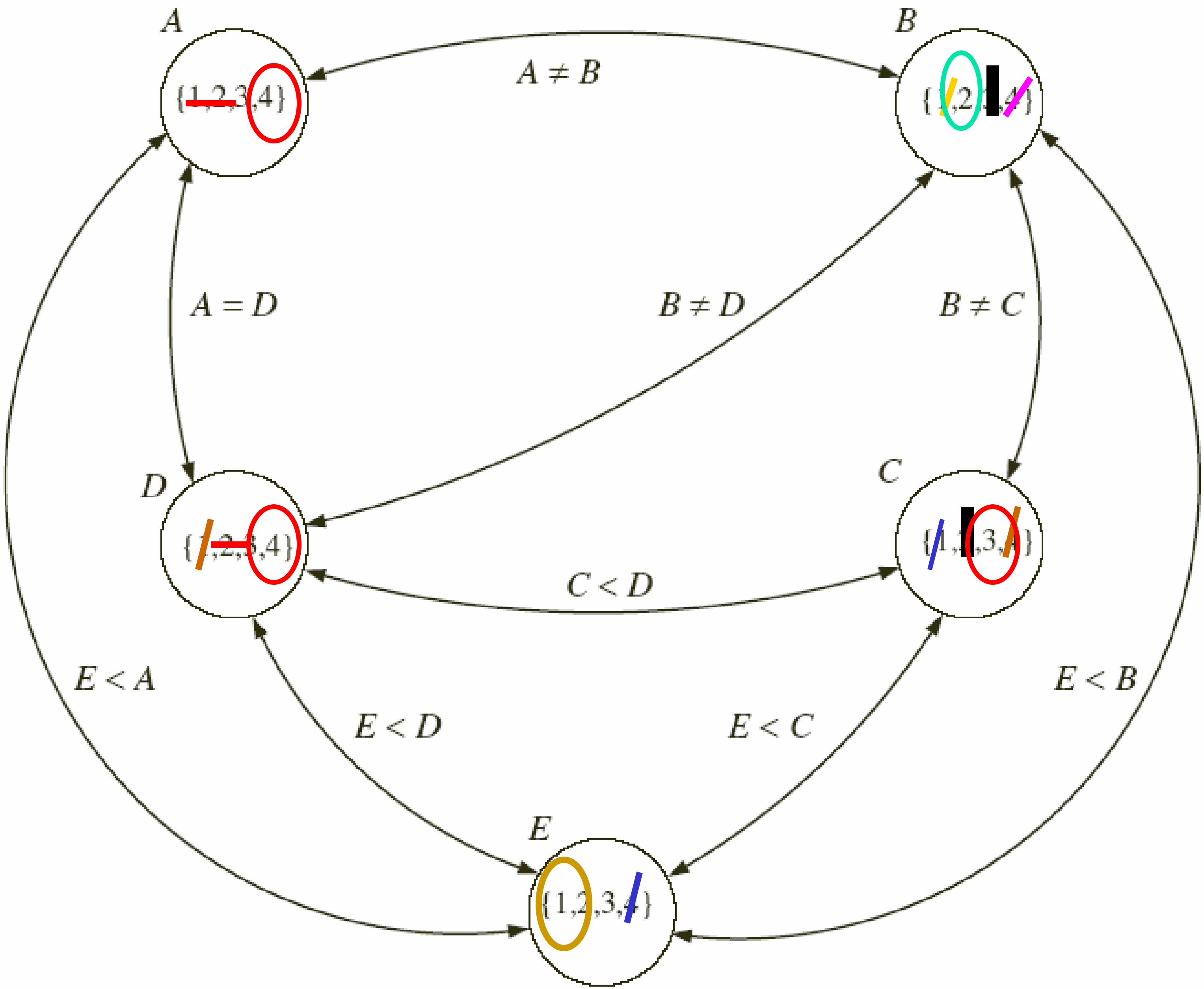
  if

  $\forall x \in D_X \; \exists y \in D_Y$ s.t. $\langle x,y \rangle \in C_{X,Y}$

- Eg: $D_A = \{1, 2, 3, 4\}$ and $D_E = \{1, 2, 3, 4\}$
  NOT arc consistent as

  $A = 1$ is not consistent with $E < A$

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



E < C

C < D

⇒ C=3

⇒ D=4

⇒ A=4

E<B

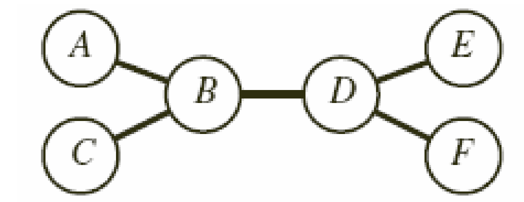
A≠B

⇒ B=2

⇒ E=1

Note: Already removed B = 3, C = 2

Special Case: Tree structured CSPs

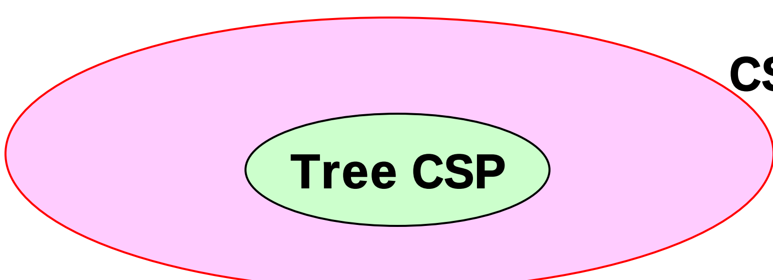


- *Theorem*: If the constraint graph is tree-structured (has no loops),
  Arc-Consistency is sufficient!
  $\Rightarrow$ CSP can be solved in $O(n\ |D|^2)$ time.

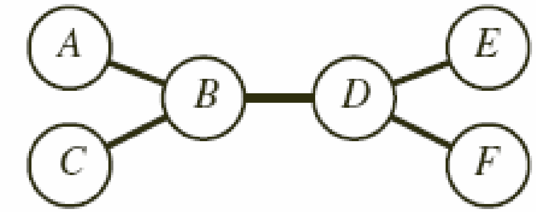
- For general CSPs: worst-case time is $O(|D|^n)$

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

CSP = Binary CSP



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

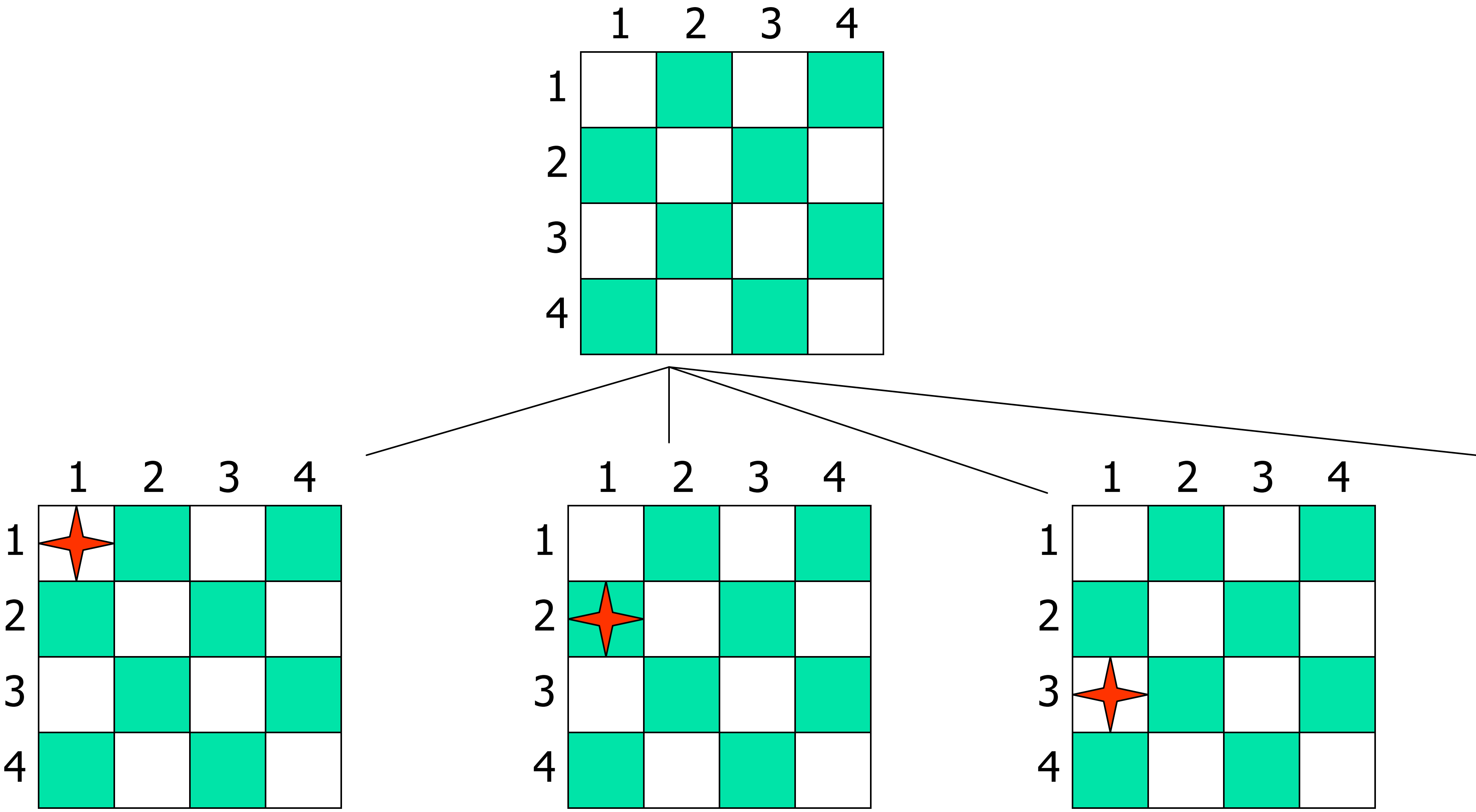
Just Arc Consistency is enough… think 2SAT!

Additional Propagation

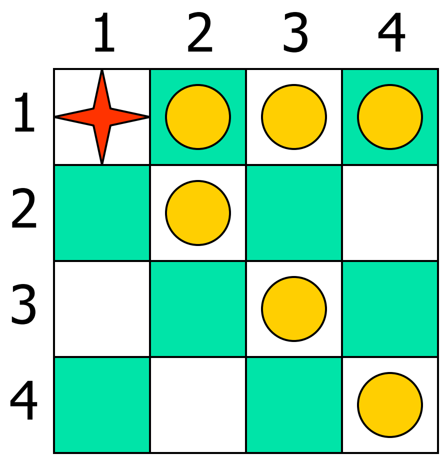
- More elaborate propagation, *DURING computation*:

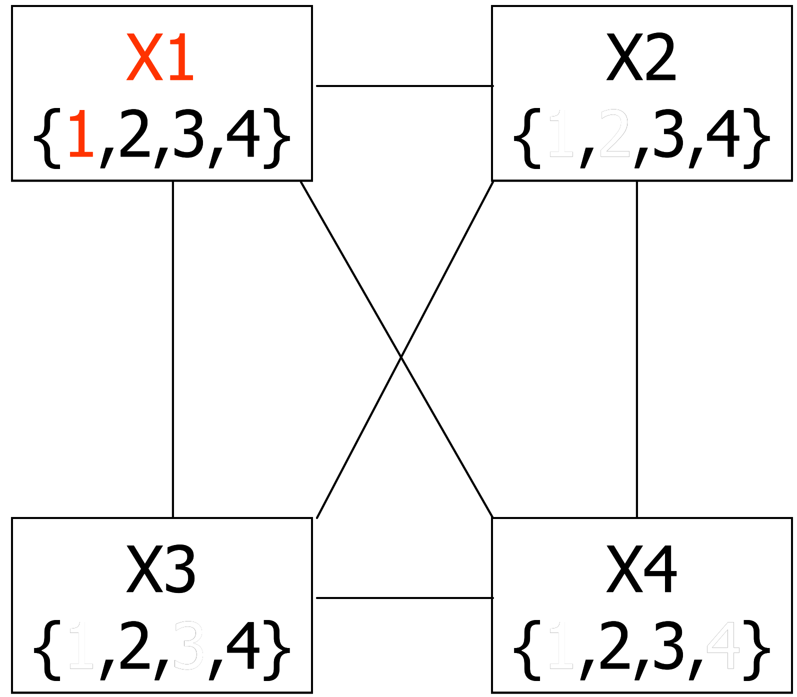
  - Assign
  - Propagate

- Assign $X_i := v$, then propagate effects to future variables
  - Whenever remove value from X_j, consider effects wrt X_j's neighbors…


1 2 3 4
1
2
3
4
1 2 3 4
1
2
3
4
1 2 3 4
1
2
3
4
1 2 3 4
1
2
3
4

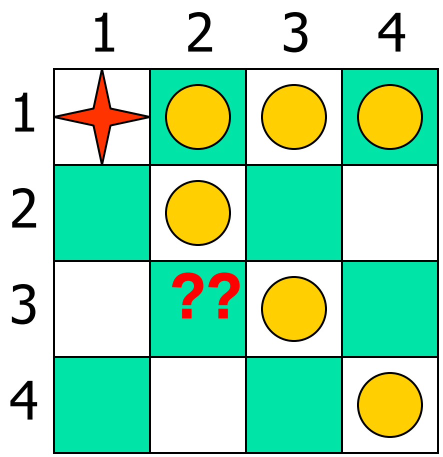
4-Queens Problem

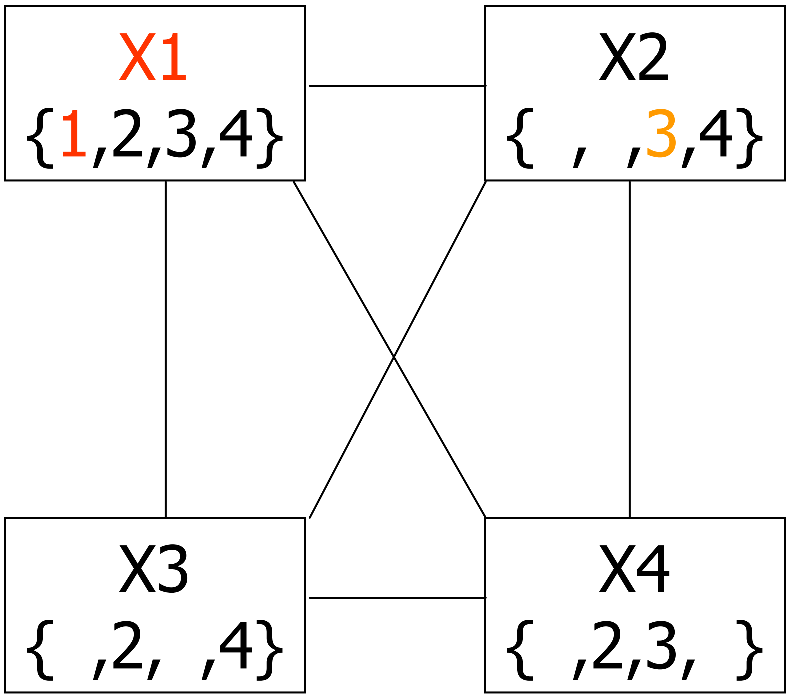




4-Queens Problem

Now run "Make Arc Consistent"
… Constraint Propagation …

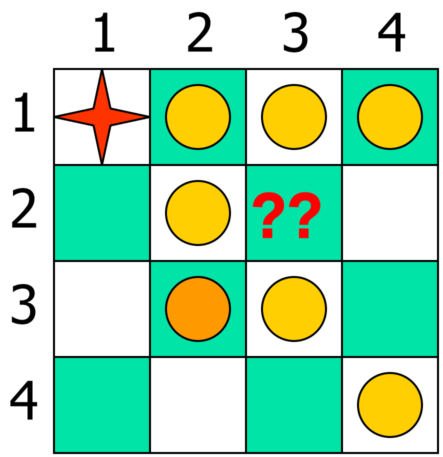


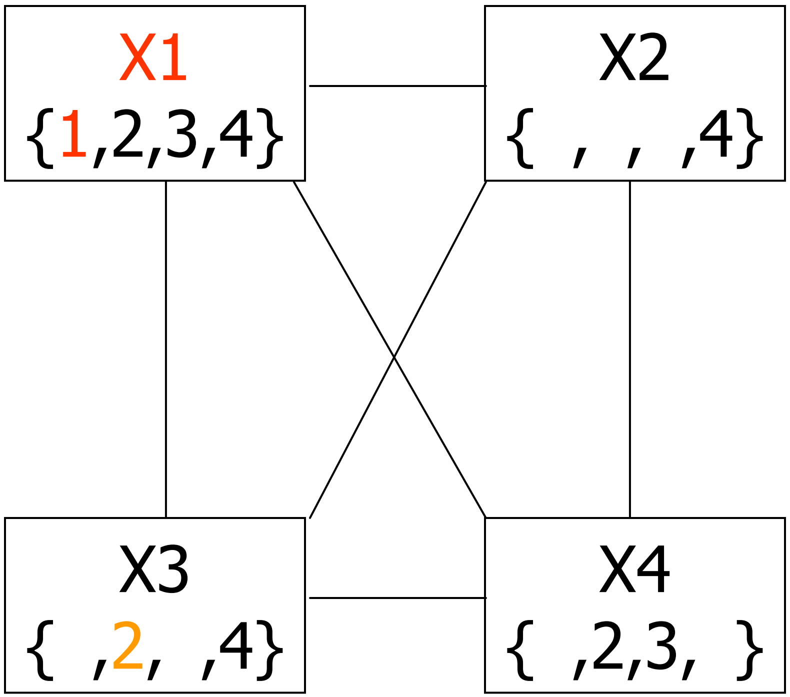


X2=3 is *not consistent* with any remaining value of X3 $\in$ {2, 4} $\Rightarrow$ REMOVE X2=3 !

4-Queens Problem

Now run "Make Arc Consistent"
... Constraint Propagation ...

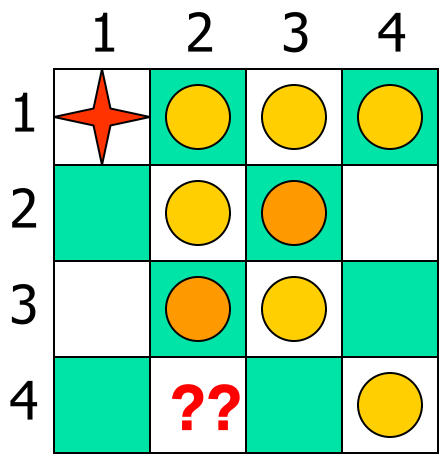


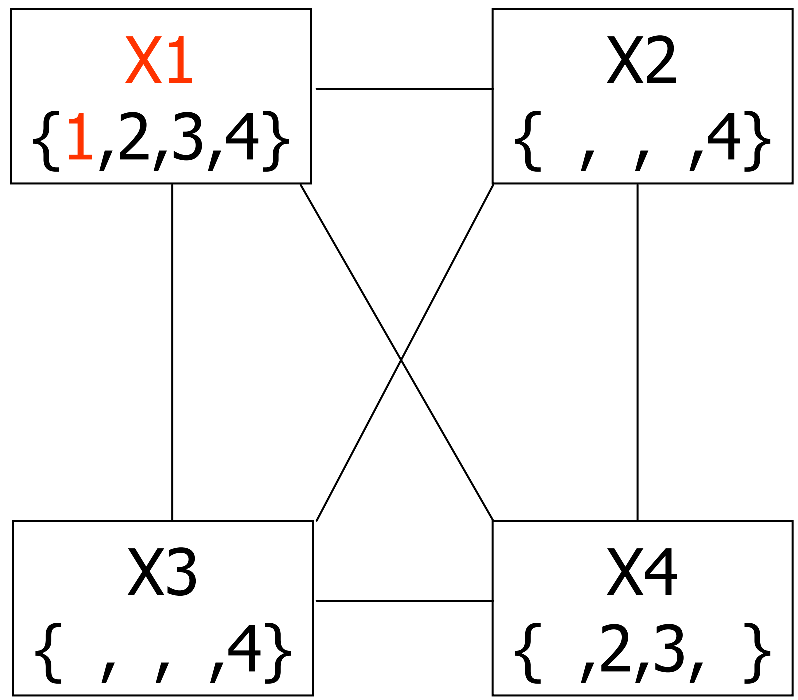


X3=2 is *not consistent* with any remaining value of X4 $\in$ {2, 3} $\Rightarrow$ REMOVE X3=2 !

4-Queens Problem

Now run "Make Arc Consistent"
... Constraint Propagation ...

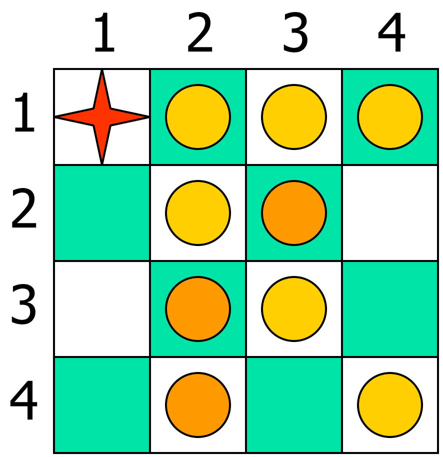


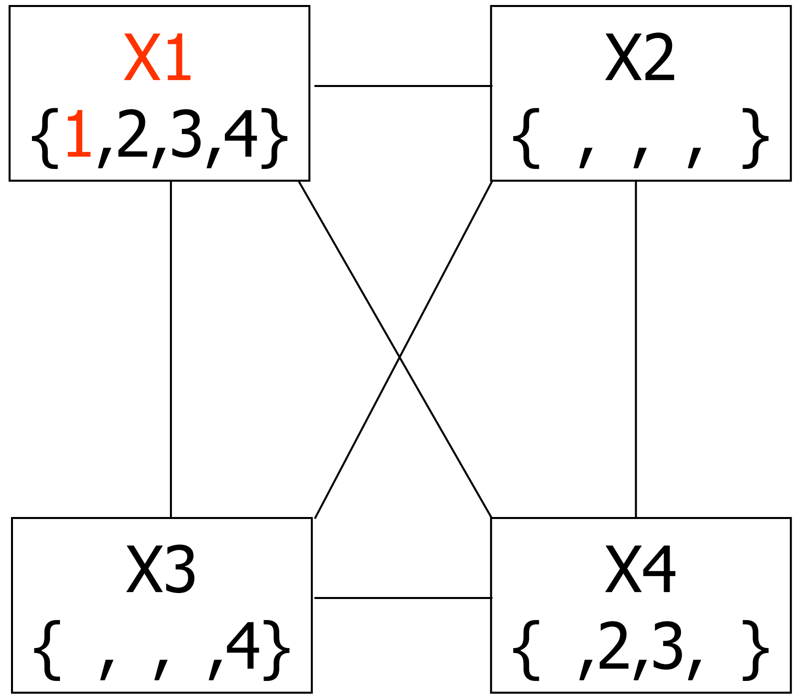


X2=4 is *not consistent* with any remaining value of X3 $\in$ {4} $\Rightarrow$ REMOVE X2=4 !

4-Queens Problem

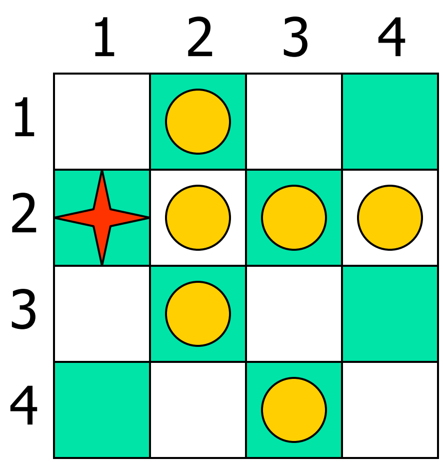
Now run "Make Arc Consistent"
... Constraint Propagation ...

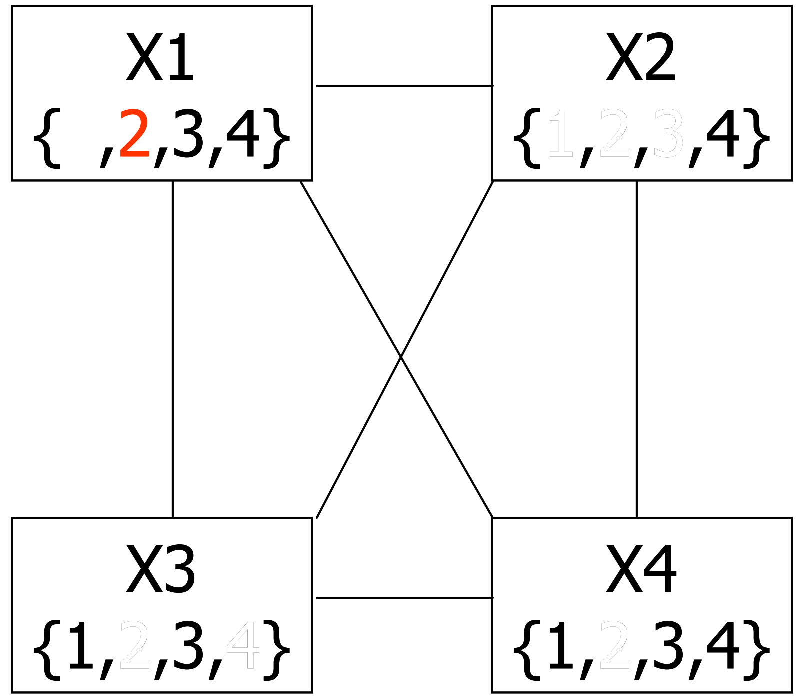




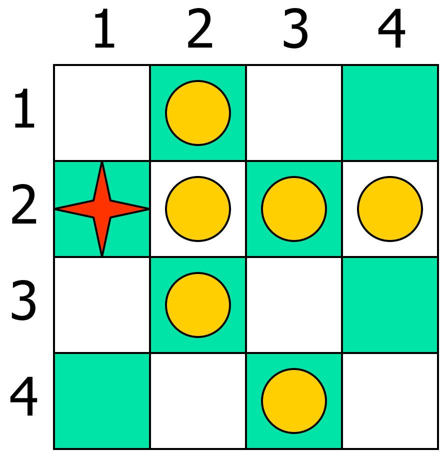
No value for X2, so backtrack!

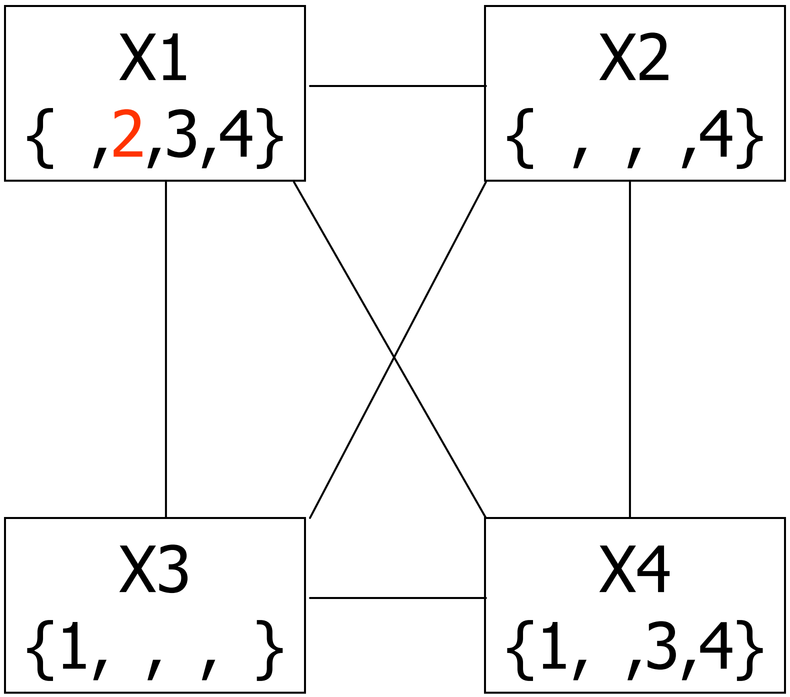
4-Queens Problem



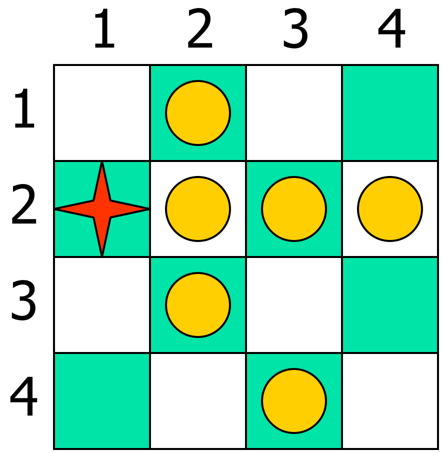


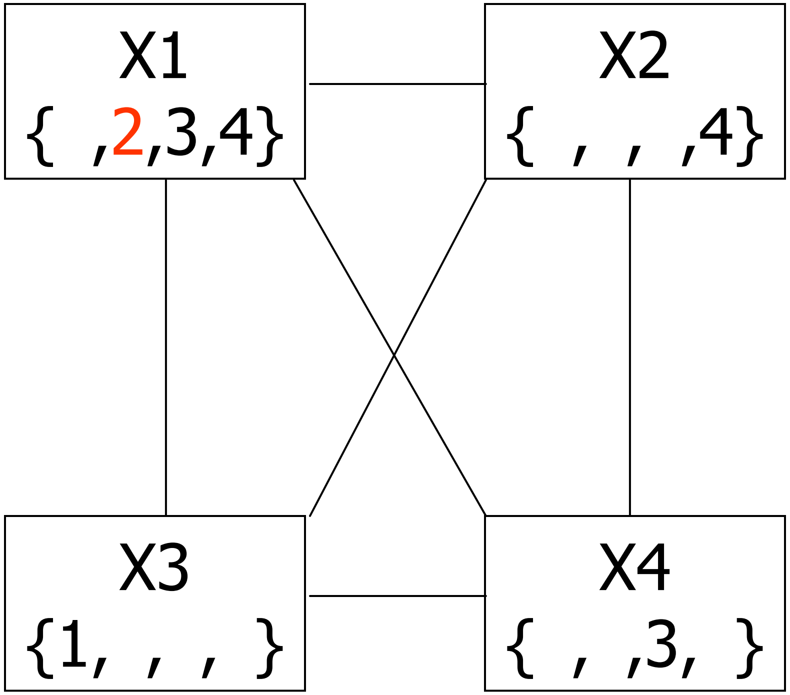
4-Queens Problem



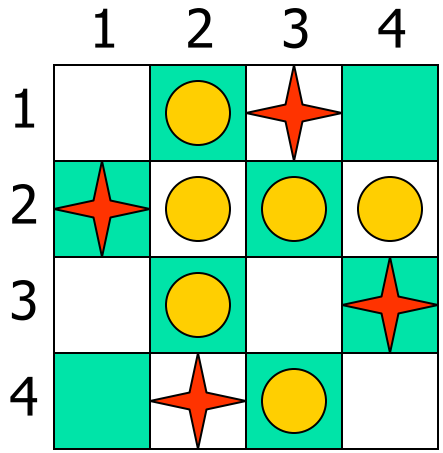


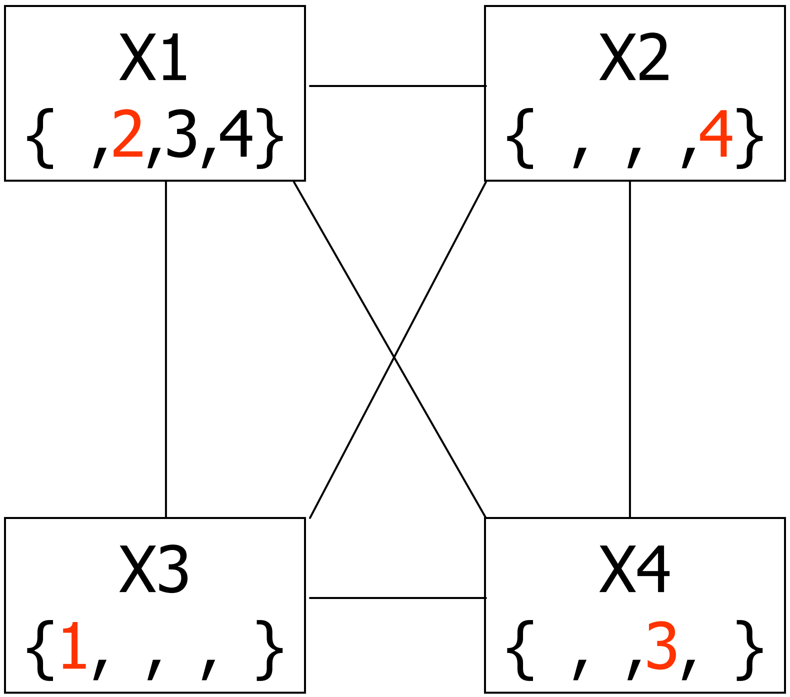
4-Queens Problem





4-Queens Problem





General CP for Binary Constraints … MakeArcConsistent

MAC (variables, constraints): Boolean

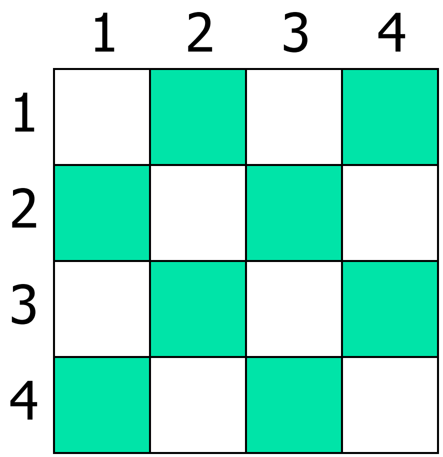
- contradiction ← *false*
- Q ← stack of all variables
- while Q is not empty and not contradiction do
  - X ← UnSTACK(Q)
  - For every variable Y adjacent to X do
    - If REMOVE-ARC-INCONSISTENCIES(X,Y) then
      - If Y's domain is non-empty
        Then STACK(Y, Q)
        Else return *false*

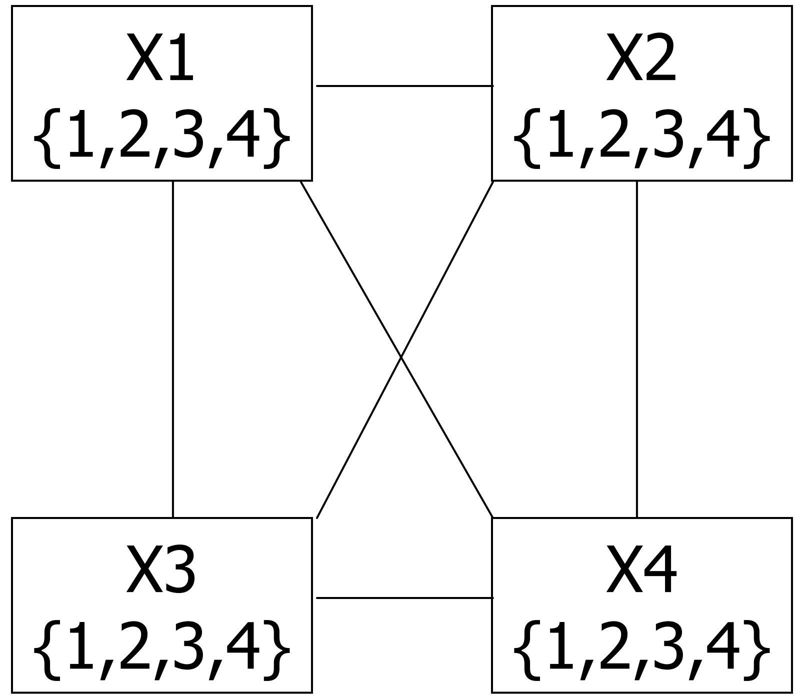
Also called AC3

Complexity Analysis of MAC

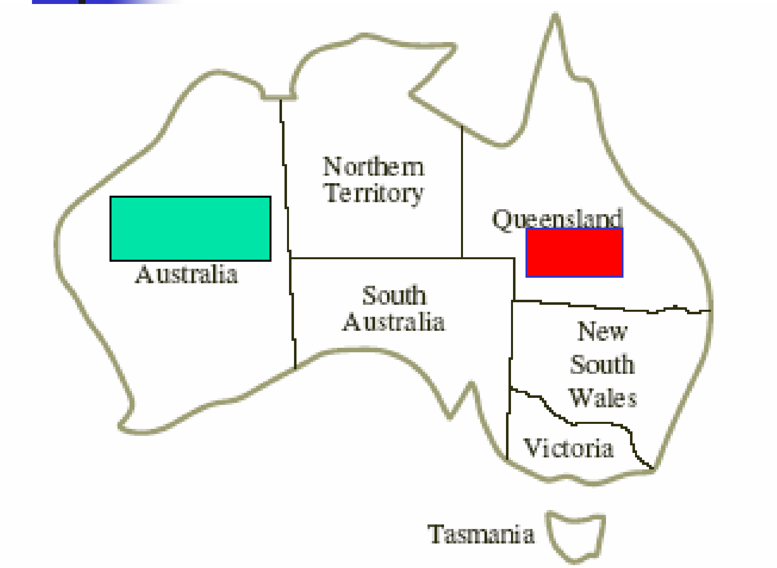
- e = number of constraints (edges)
- d = number of values per variable
- Each variable is inserted in Q $\leq d$ times
- REMOVE-ARC-INCONSISTENCY takes $O(d^2)$ time
- MAC takes $O(ed^3)$ time

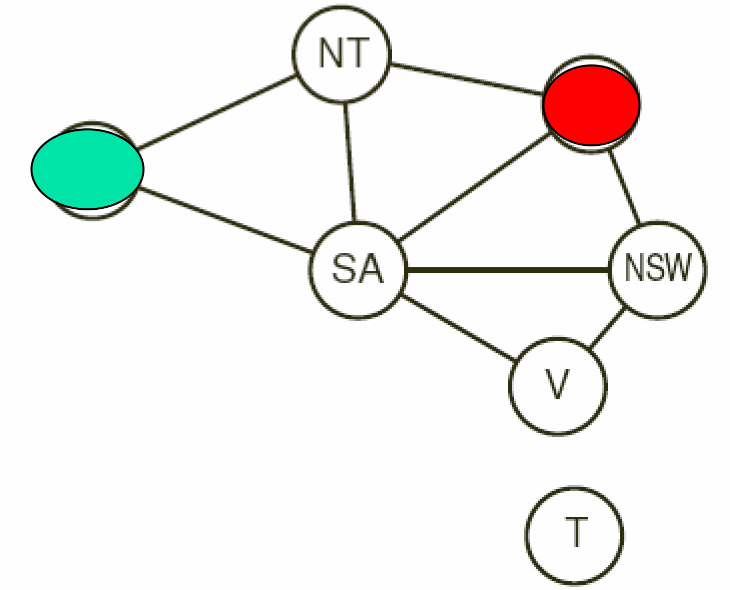
Is MAC Alone Sufficient?





Does Assign+MAC solve everything?





- After MAC…
  - domain for NT = { B }
  - domain for SA = { B }
  - … but NT ≠ SA!!

Trick#4: Best Variable/Value

4a. Most-constrained variable first:

- Select unassigned variable with smallest domain
- Dynamic: after each pruning w/forward checking, ...
- Eg: If $|D_E| = 2$ and $|D_i| \geq 3$ for other i, select E

4b. Most-constraining variable first:

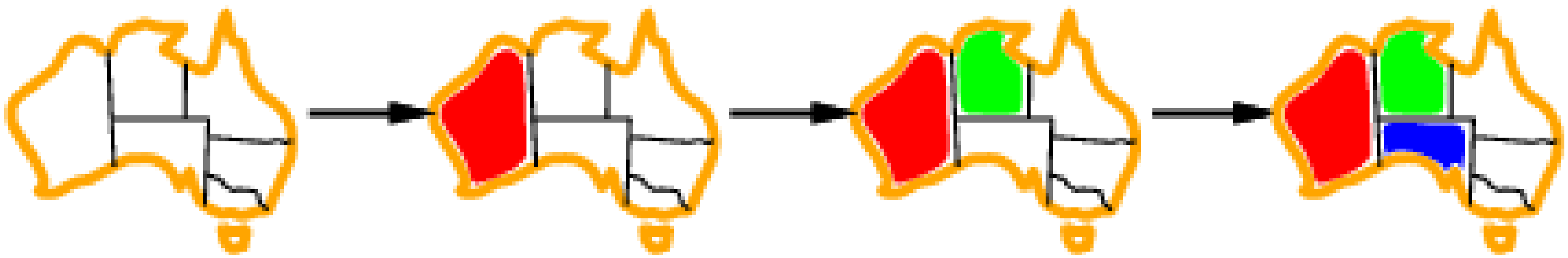
- Select unassigned variable that appears in most constraints w/ other unassigned variables
- Let $f(X) = |\{ Y : Y \text{ unassigned}; \exists C_{\ldots X \ldots Y} \}|$
  Select $X^* = \arg\min_X \{ f(X) : X \text{ unassigned} \}$
- Eg: Start with B, as $f(B) = 4 \geq f(X)\ \forall X$, ...

4c. Least-constraining value first:

- Choose value for X that leaves the most values for OTHER unassigned variables

4a: Most Constrained Variable

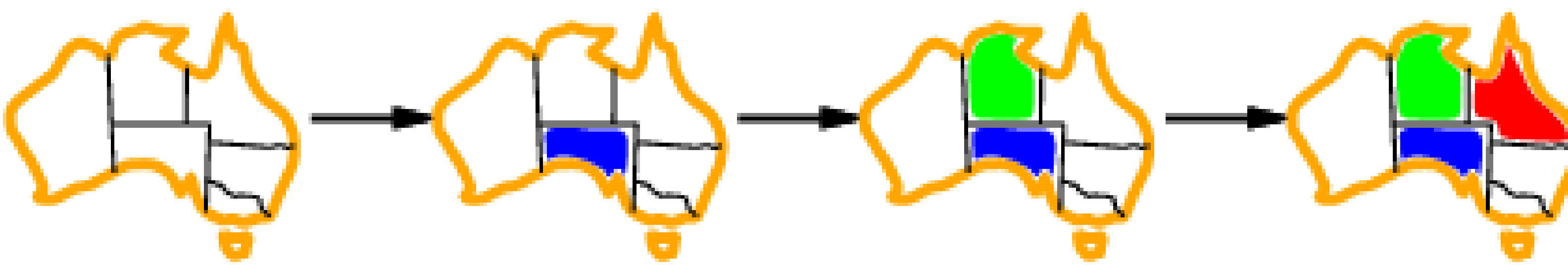
- Most constrained variable:
  choose the variable with the fewest legal values



- a.k.a. minimum remaining values (MRV)
- If going to fail, FAIL QUICKLY!

4b: Most Constraining Variable

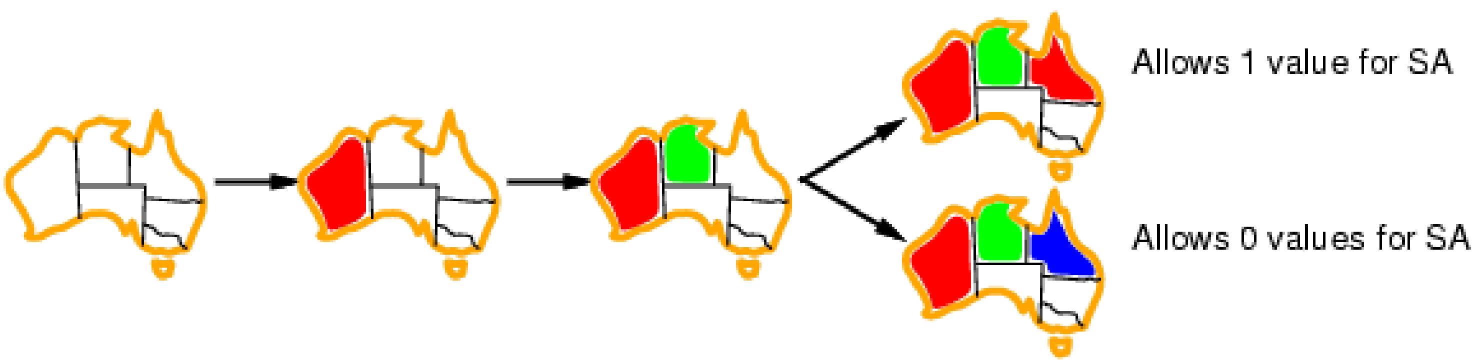
- Most constraining variable:
  - choose the variable involved in largest # of constraints on remaining variables



- Tie-breaker among most constrained variables

4c: Least constraining value

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables



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
  - with 10,000 to 100,000 variables

Hard CSPs

- Suppose all constraints UNARY (explicit)

$\Rightarrow$ 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
  - # of rooms required
  - # chip size
  - # time for delivery
- 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 )
```

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. **satisfiability problems:** satisfies given set of "hard" constraints
     or
  2. **optimization problems** (**"**soft constraints")
     minimizes given cost function,
       where each assignment to variables has cost
- **In general**,
  + different domains for different 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

Constraint Satisfaction Problems

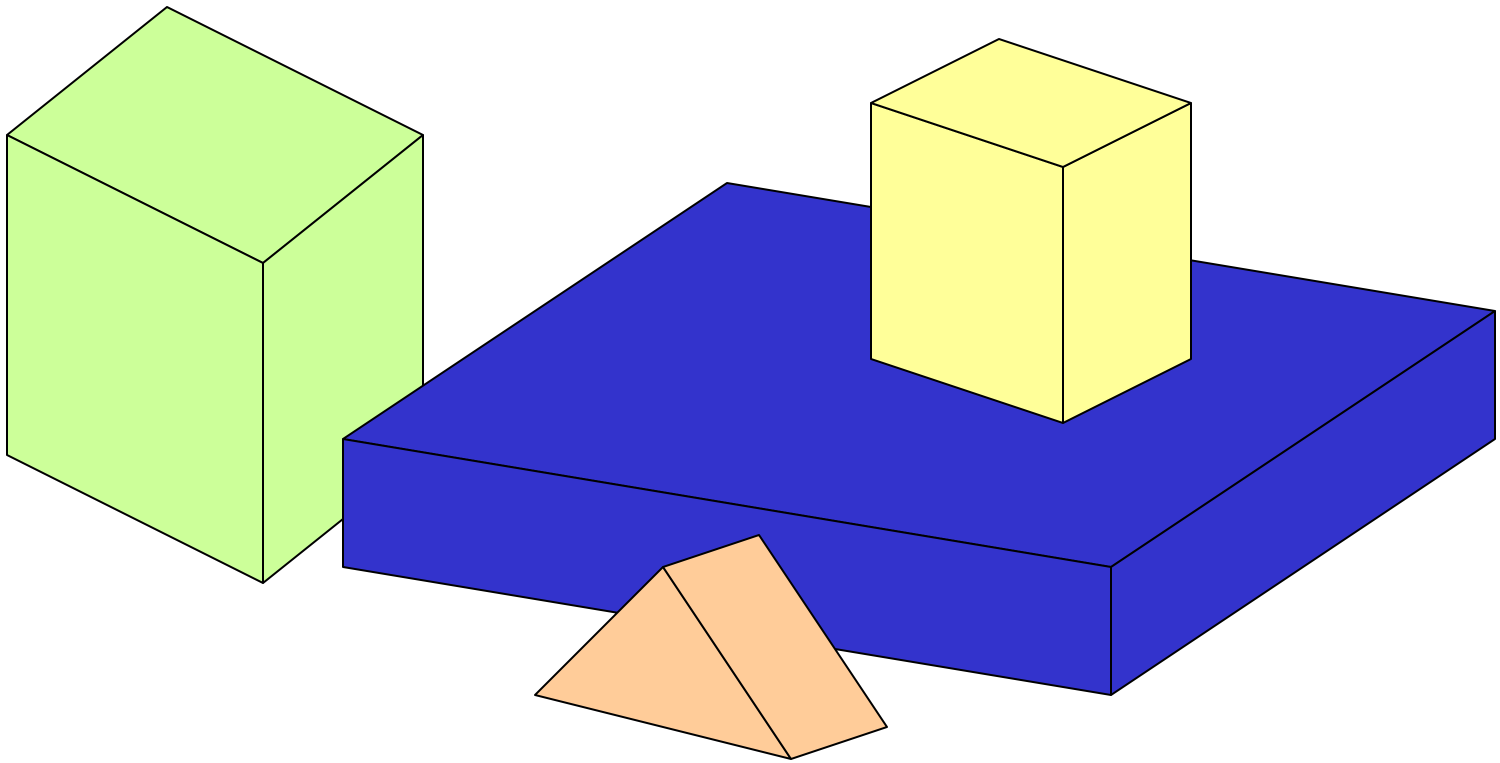
- Scheduling Courses: Assign time/prof/room to each course
  - "Hard Constraints" (requirements)
    + Prof can only be at one place at any time
    + Course + Lab must be at dierent times
    + Only one course to a room, . . .
  - "Soft Constraints" (preferences)
    + Companion classes should be close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

Edge Labeling in Computer Vision

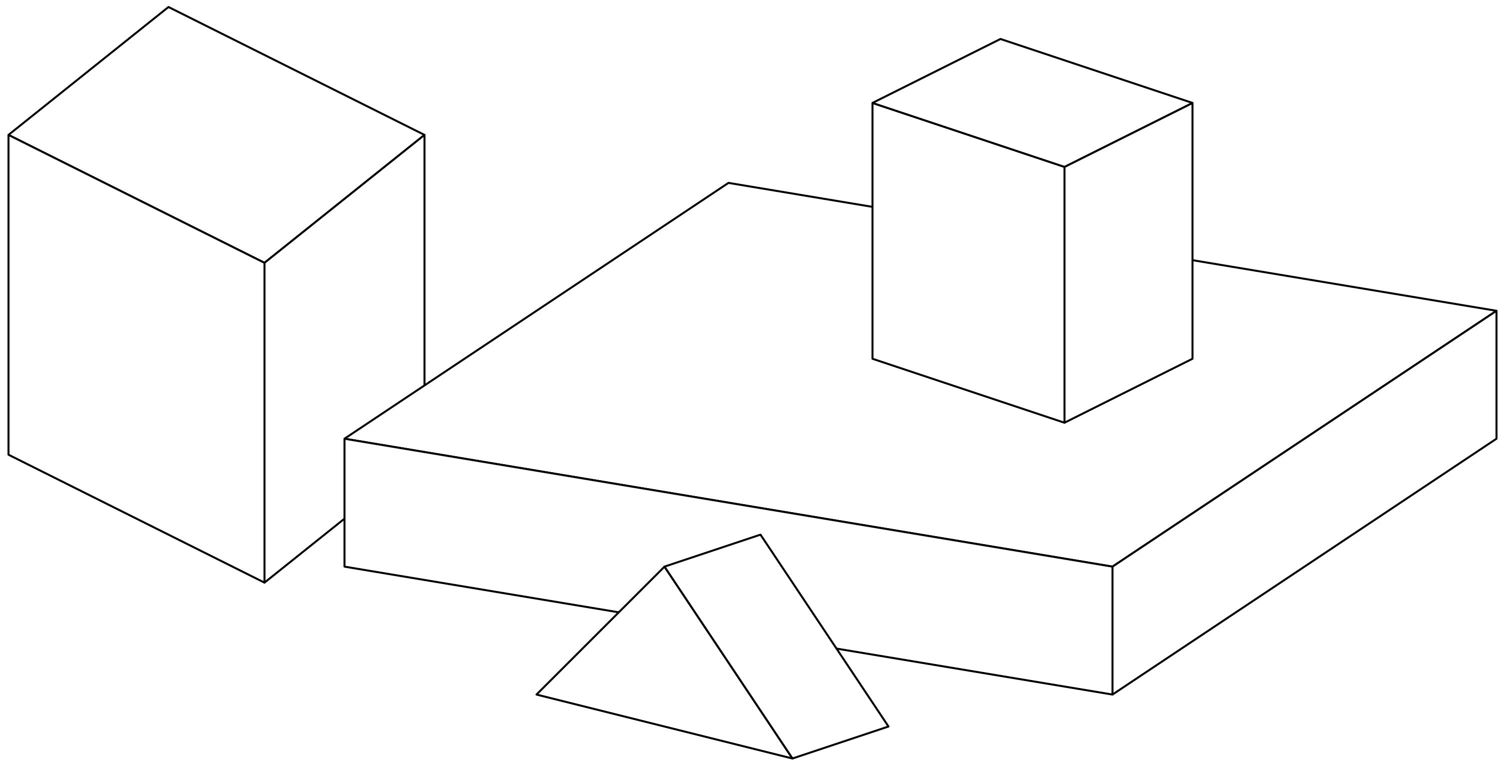
Russell and Norvig:
Chapter 24, pages 745-749

Skip to end...

Edge Labeling



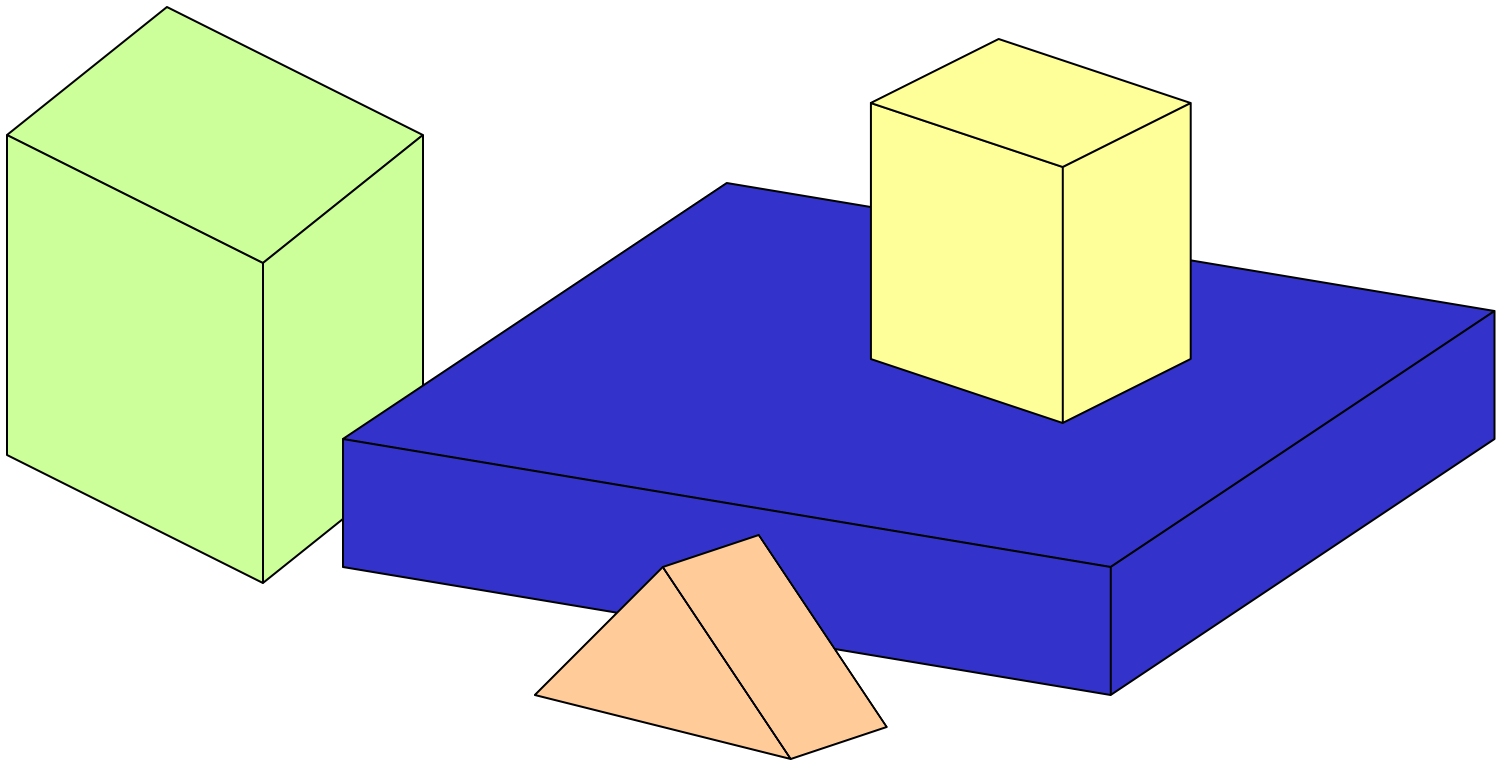
Edge Labeling



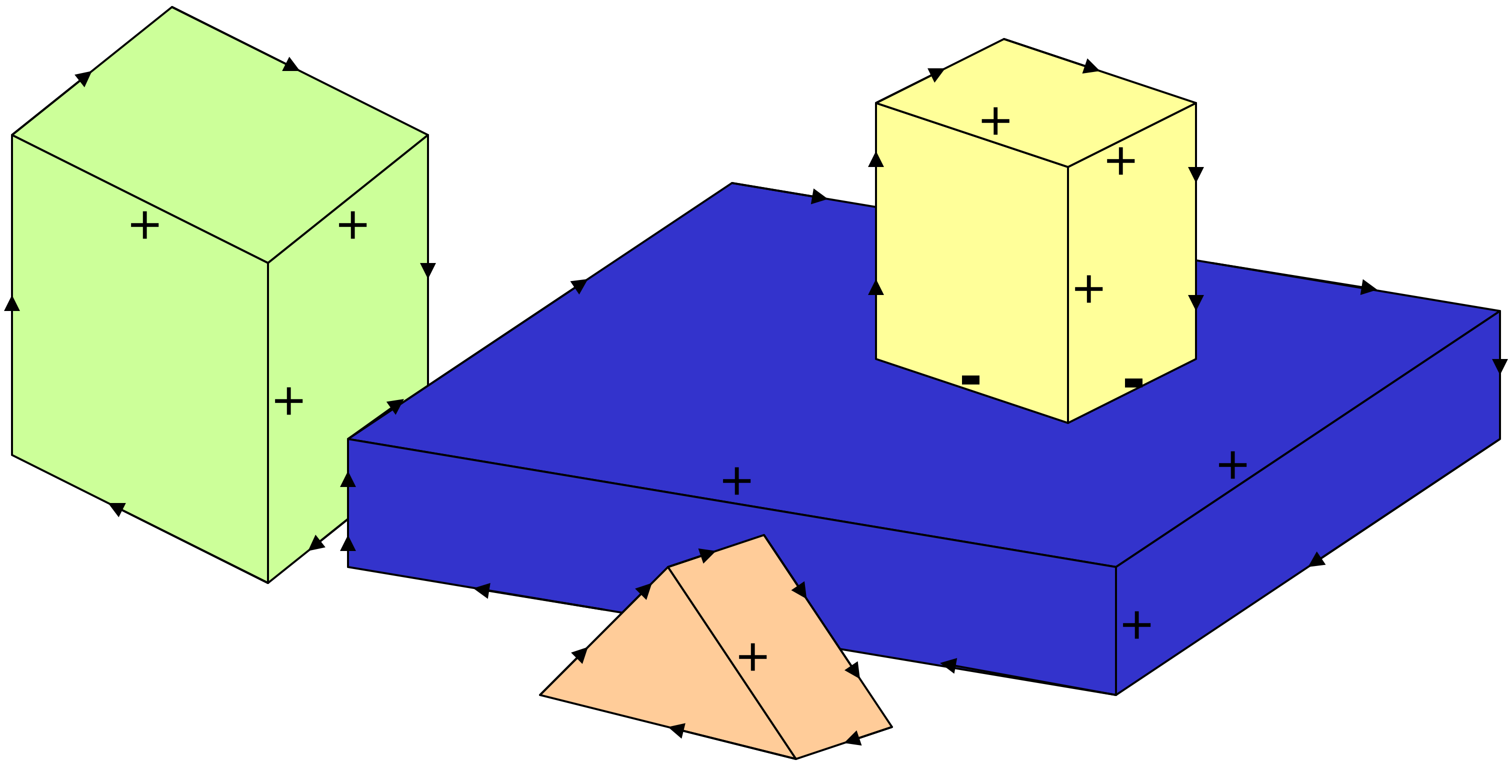
Labels of Edges

- Convex edge:
  - two surfaces intersecting at an angle greater than 180°
- Concave edge
  - two surfaces intersecting at an angle less than 180°
- + convex edge, both surfaces visible
- − concave edge, both surfaces visible
- ← convex edge, only one surface is visible and it is on the right side of ←

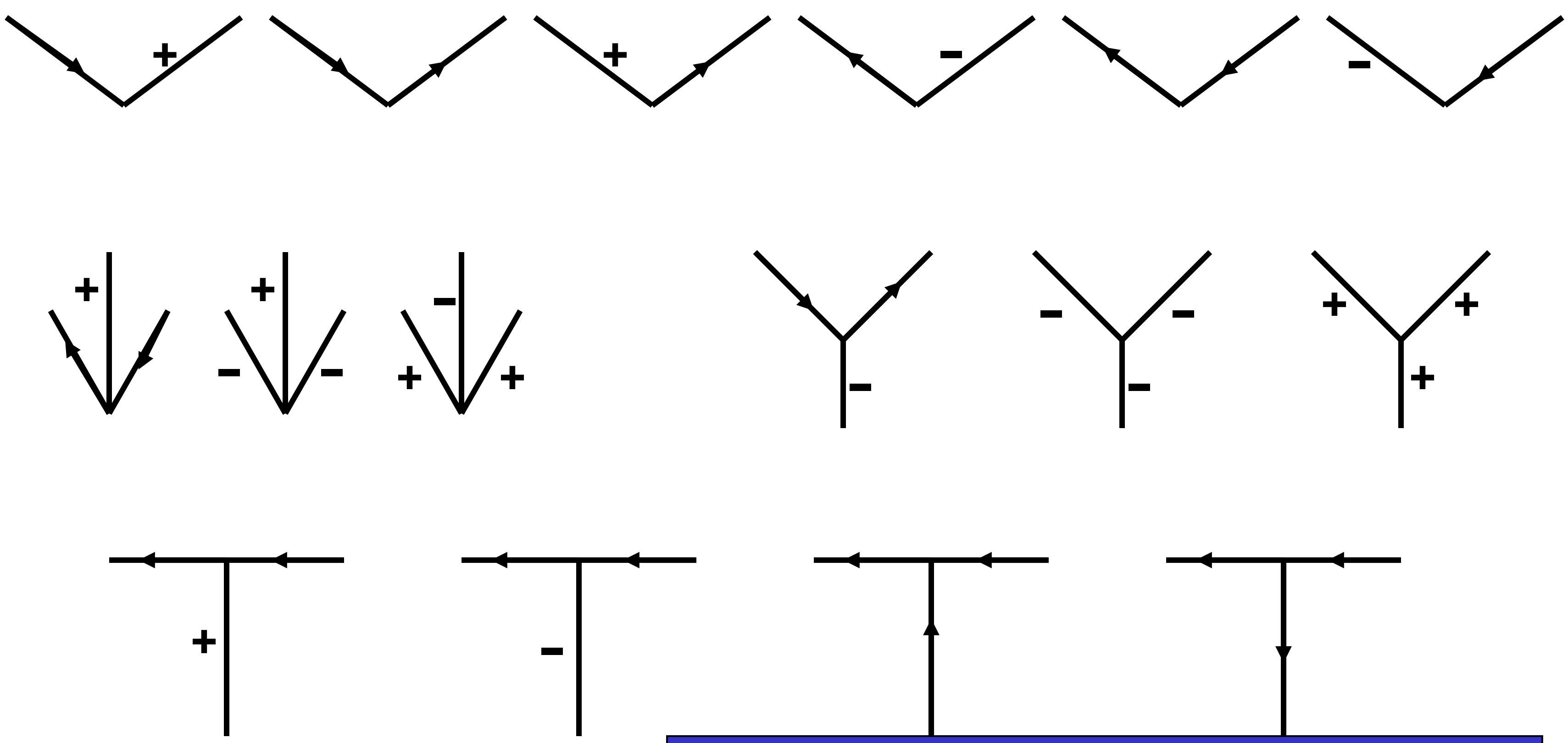
Edge Labeling



Edge Labeling



Junction Label Sets

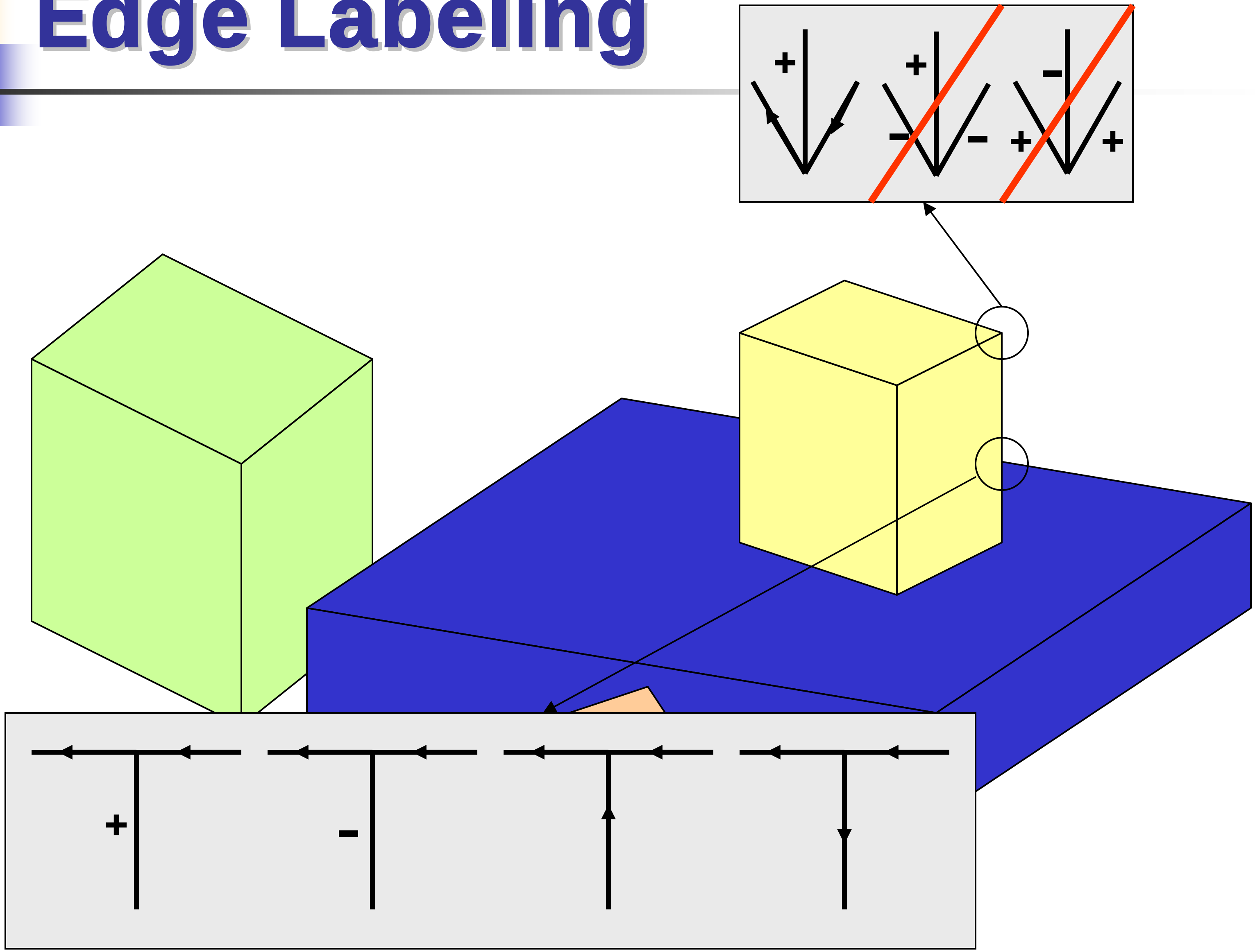


(Waltz, 1975; Mackworth, 1977)

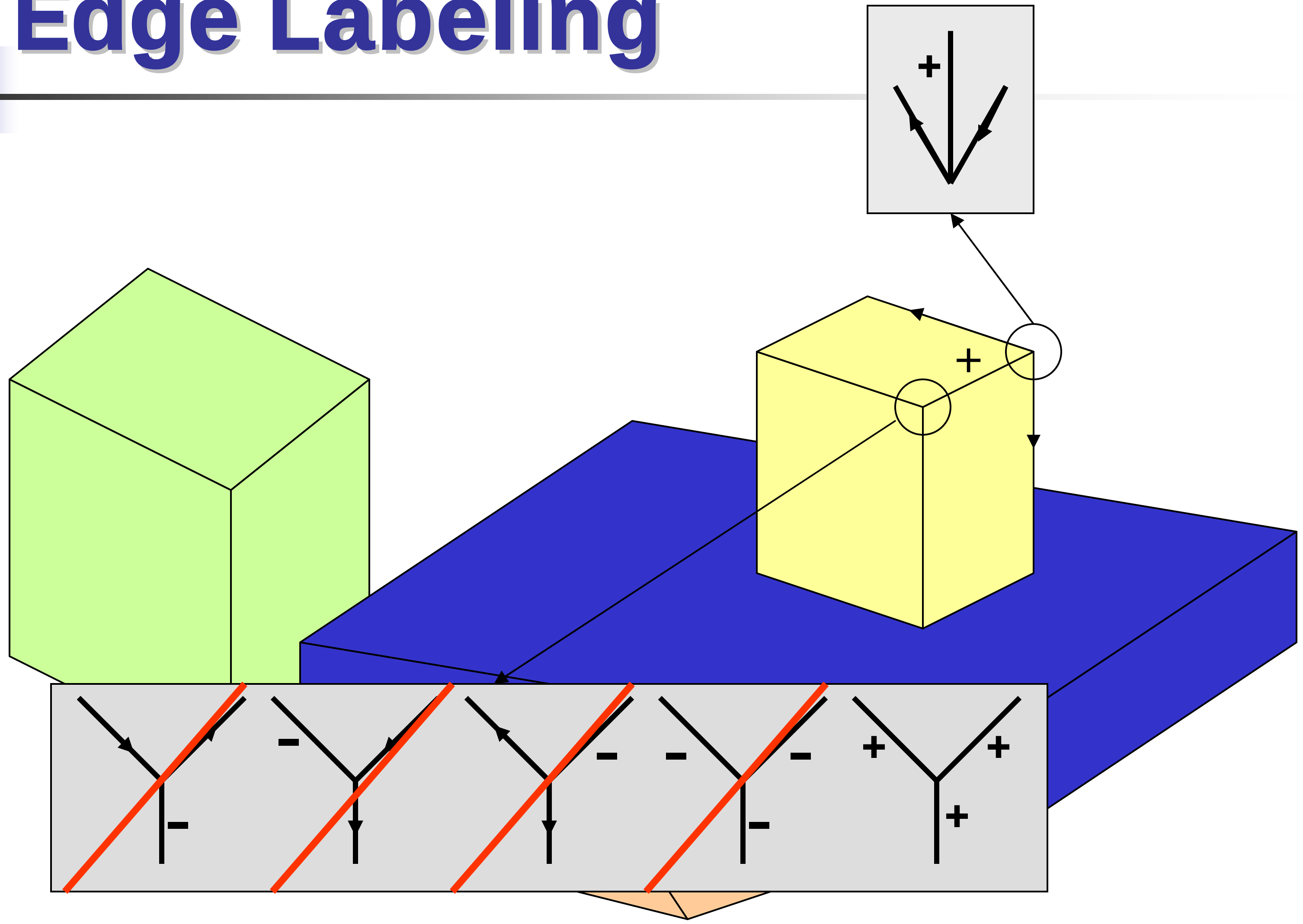
Edge Labeling as a CSP

- Variable associated with each junction
- Domain of a variable =
  the label set of the corresponding junction
- Each constraint states
  the values given to two adjacent junctions
  give the same label to the joining edge

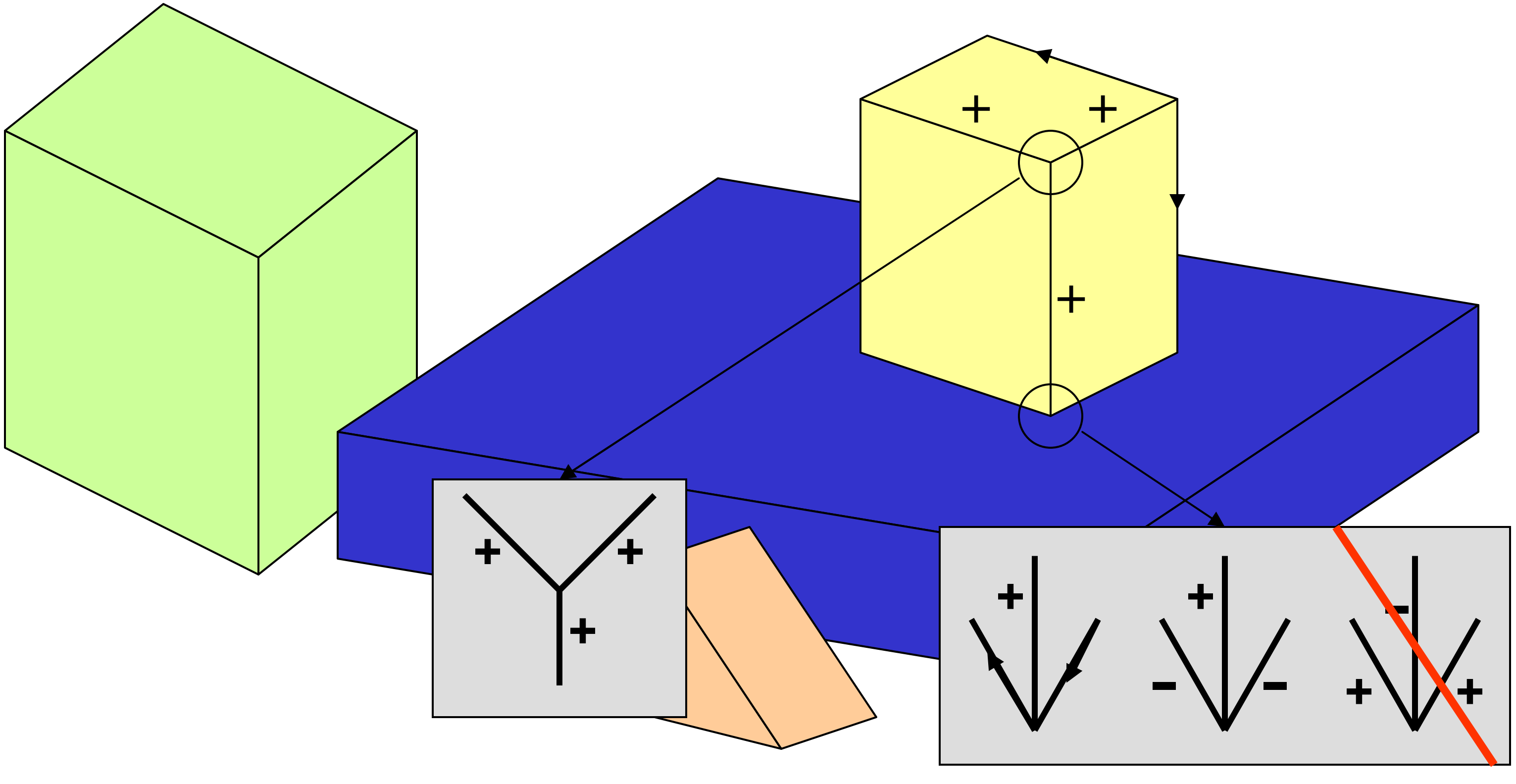
Edge Labeling



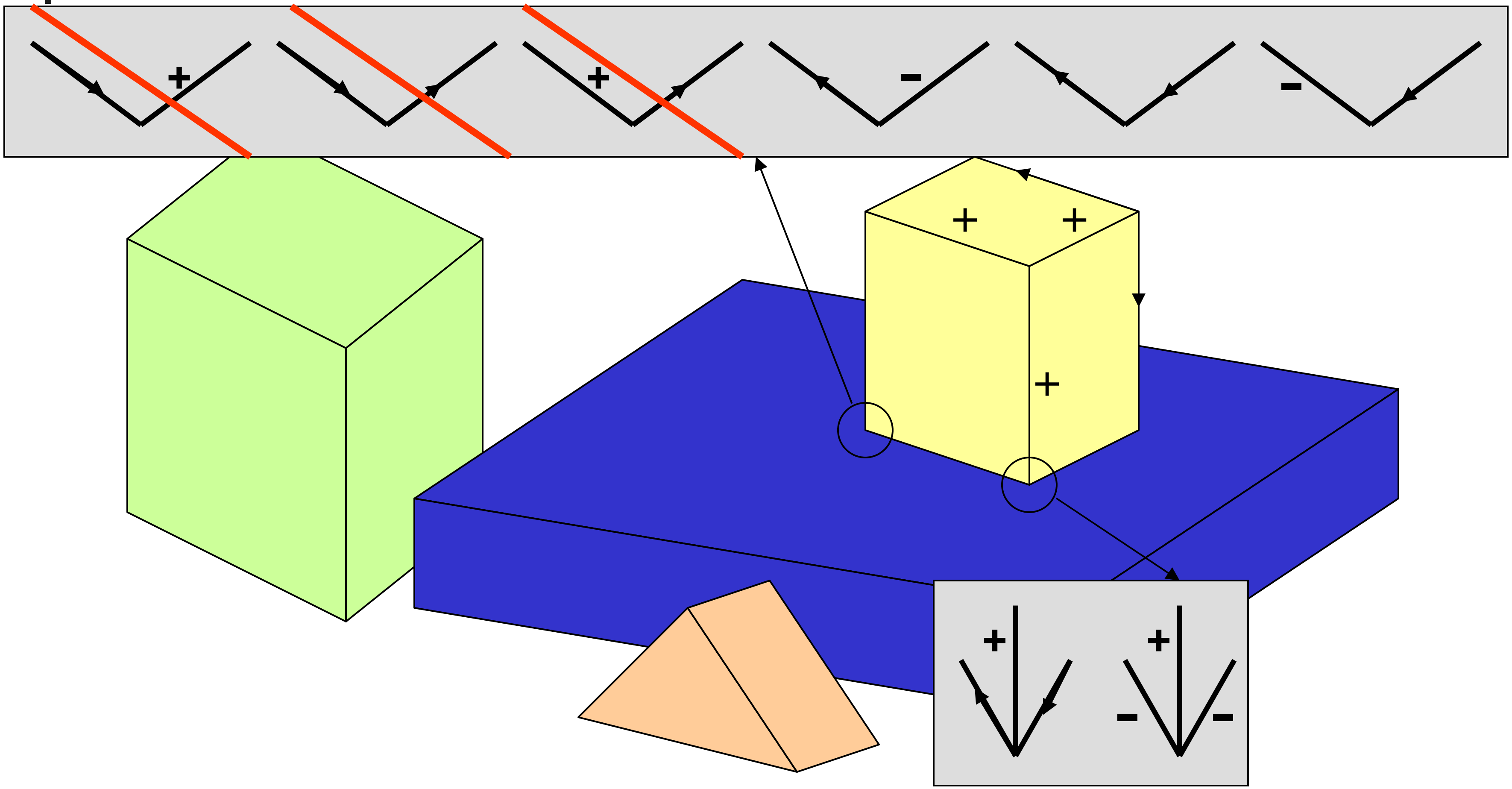
Edge Labeling



Edge Labeling



Edge Labeling



Comments

- Why not Mathematical Programming Problem?
  - CSP rep'n more natural/expressive
    - + variables problem entities
    - + constraints natural description
    - (not just linear inequalities)
  - $\Rightarrow$ Formulation simpler, solution easier to understand, easier to find good heuristics
  - CSP algorithms often nd 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

Summary

- Constraint Satisfaction Problems (CSP)
- CSP as a search problem
  - Backtracking algorithm
  - General heuristics
- Forward checking
- Constraint propagation
- Edge labeling in Computer Vision