Chapter 6 Constraint Satisfaction Problems

Jane Hsu
National Taiwan University

Acknowledgements: This presentation is created by Jane hsu based on the lecture slides from *The Artificial Intelligence: A Modern Approach* by Russell & Norvig, a PowerPoint version by Min-Yen Kan, as well as various materials from the web.

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Constraint propagation algorithms
- Problem structure and decomposition
- Local search for CSPs

Example

```
    TWO
  + TWO
-----------
   FOUR
```

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

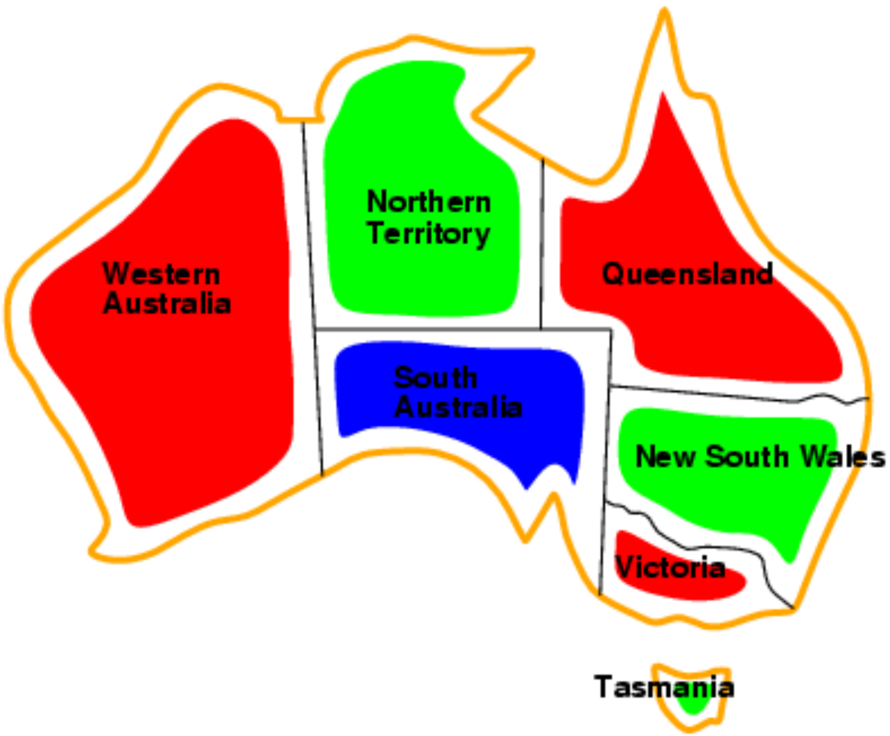
Constraint Satisfaction Problems (CSPs)

- CSP is a specialization of the general search
- state is defined by
  - variables Xi with values from domain Di
- goal test is a set of constraints specifying allowable combinations of values for subsets of variables
- Simple example of a formal representation language
- Allows useful general-purpose algorithms with more power than standard search algorithms

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)}

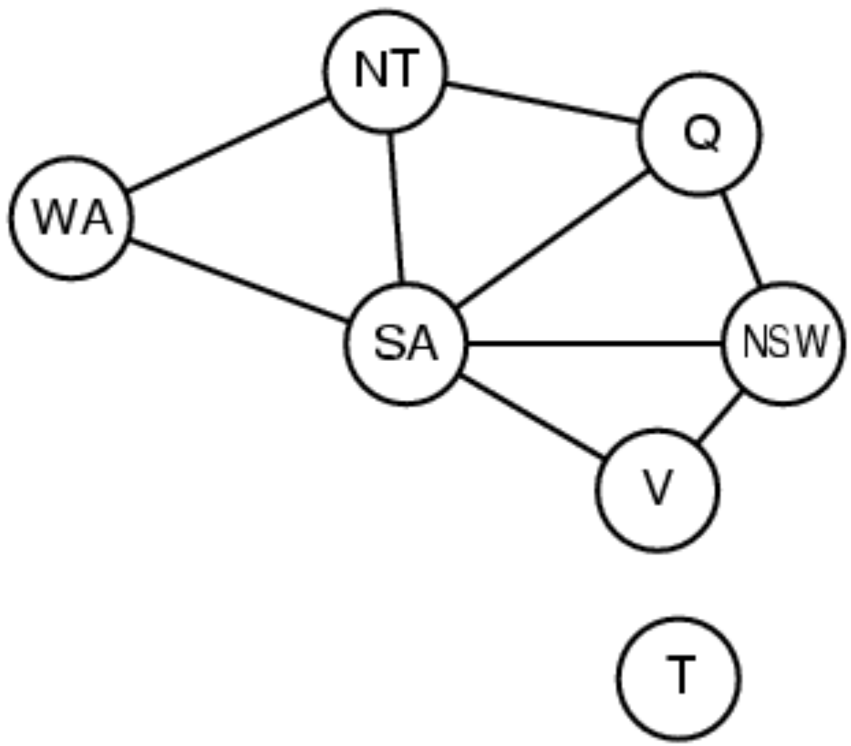
Example: Map-Coloring



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

Constraint Graph

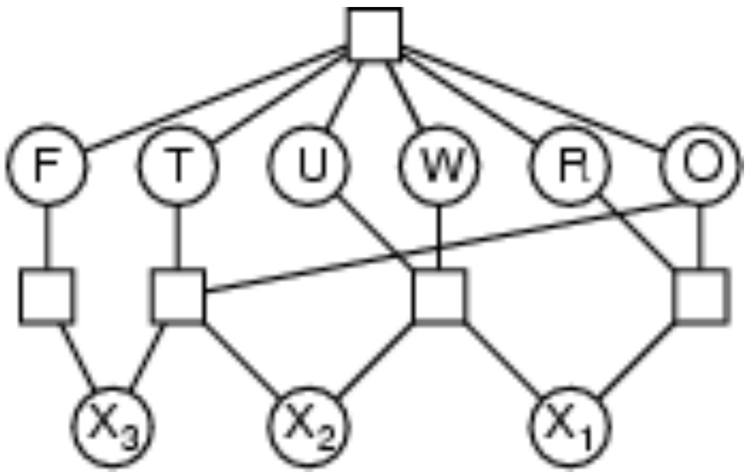
- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints



Varieties of Constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ red
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints
- Preferences (soft constraints)
  - e.g., green is better than red

Example: Cryptarithmetic



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

Varieties of CSPs

- Discrete variables
  - finite domains:
    - *n* variables, domain size *d* → $O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl. Boolean satisfiability (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

Real-World CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling

- Notice that many real-world problems may involve real-valued variables

Standard Search Formulation (Incremental)

Let's start with the straightforward approach, then fix it

States are defined by the values assigned so far

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment
  → fail if no legal assignments
- Goal test: the current assignment is complete

1. This is the same for all CSPs
2. Every solution appears at depth *n* with *n* variables
   → use depth-first search
3. Path is irrelevant, so can also use complete-state formulation
4. $b = (n - \ell)d$ at depth ℓ, hence $n! \cdot d^n$ leaves

Backtracking Search

- Variable assignments are commutative, i.e.,
  - [WA=red then NT=green] same as [NT=green then WA=red]
- Need to consider assignments to a single variable at each node
  - → Given *d* values for *n* variables, there are *dn* leaves.
- Depth-first search for CSPs with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for $n \approx 25$

Backtracking Search Algorithm

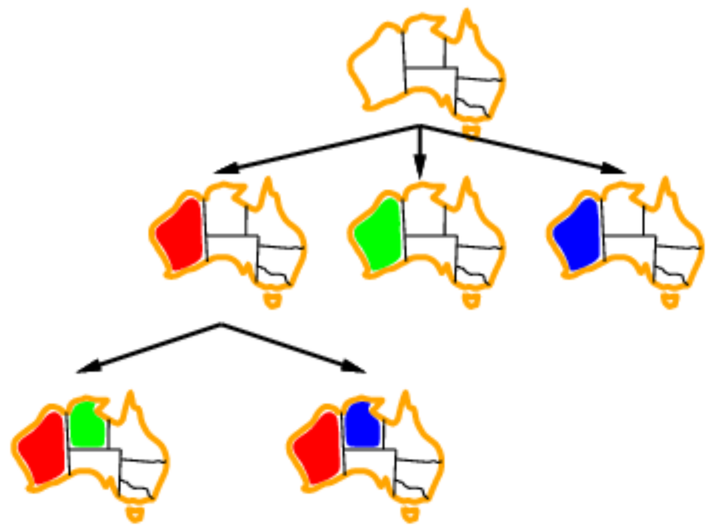
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given 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
```

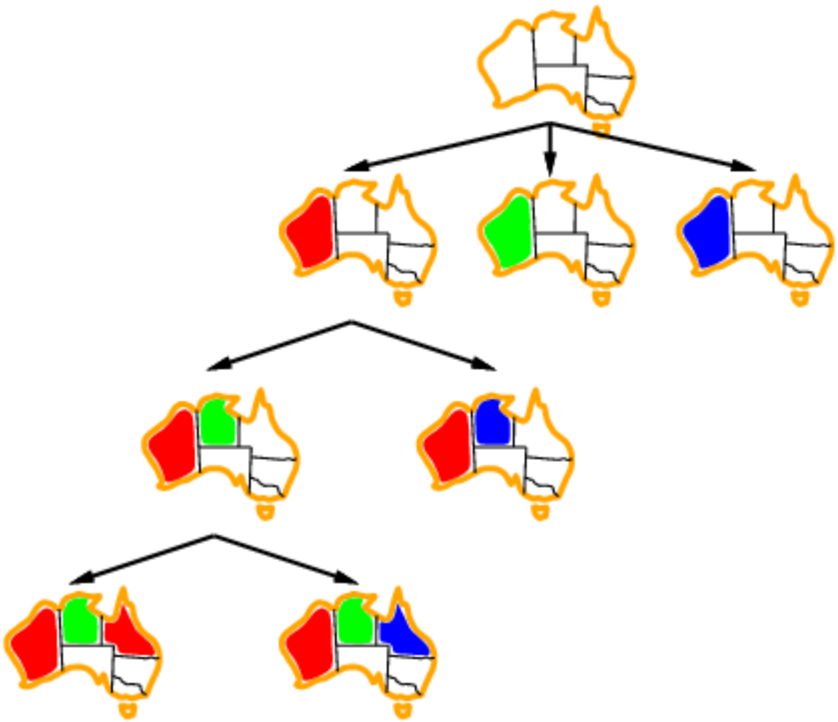
Backtracking Search Example

Backtracking Search Example

Backtracking Search Example



Backtracking Search Example

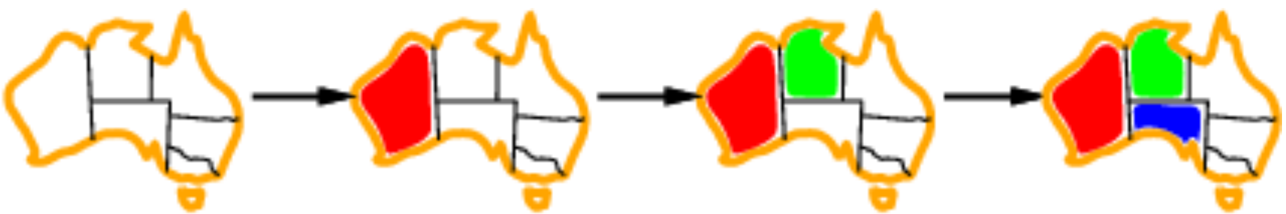


Improving Backtracking Efficiency

- General-purpose methods can give huge gains in speed:
- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failure early?

Most Constrained Variable

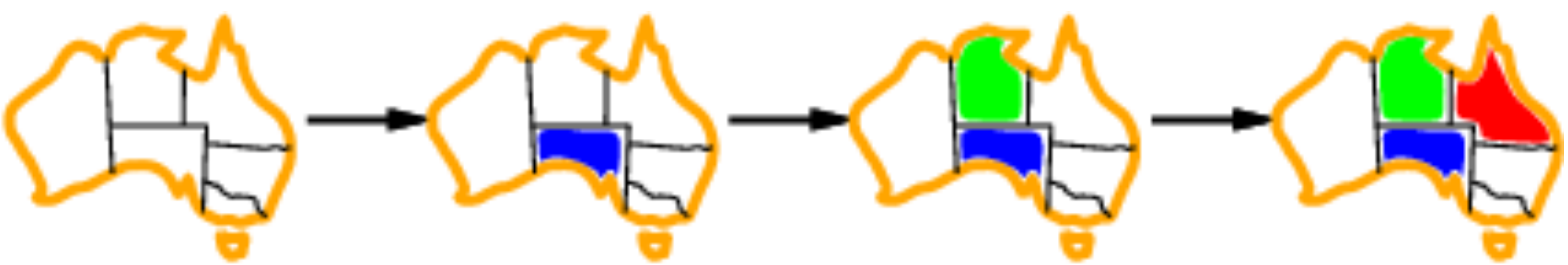
- Most constrained variable:
  choose the variable with the fewest legal values



- a.k.a. minimum remaining values (MRV) heuristic

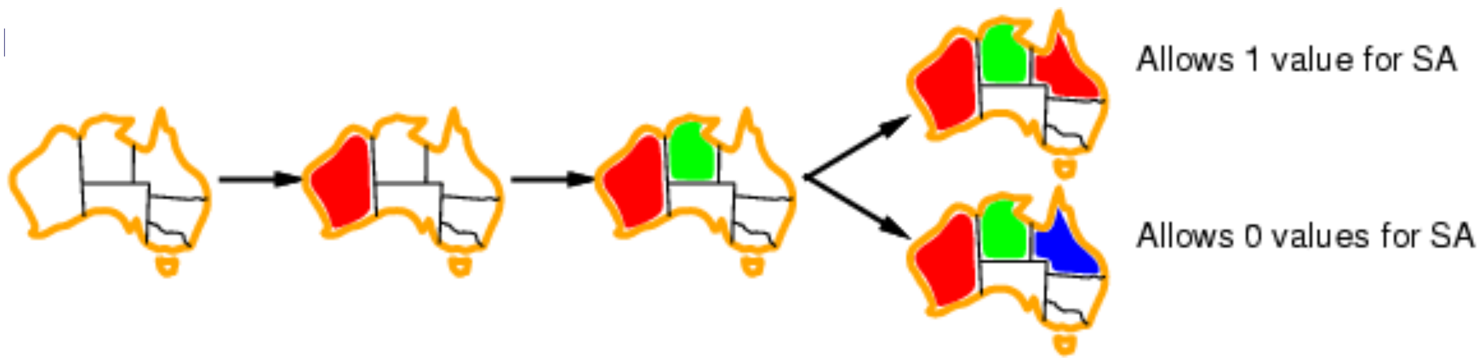
Most Constraining Variable

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables



Least Constraining Value

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



- Combining these heuristics makes 1000 queens feasible

Intelligent Backtracking

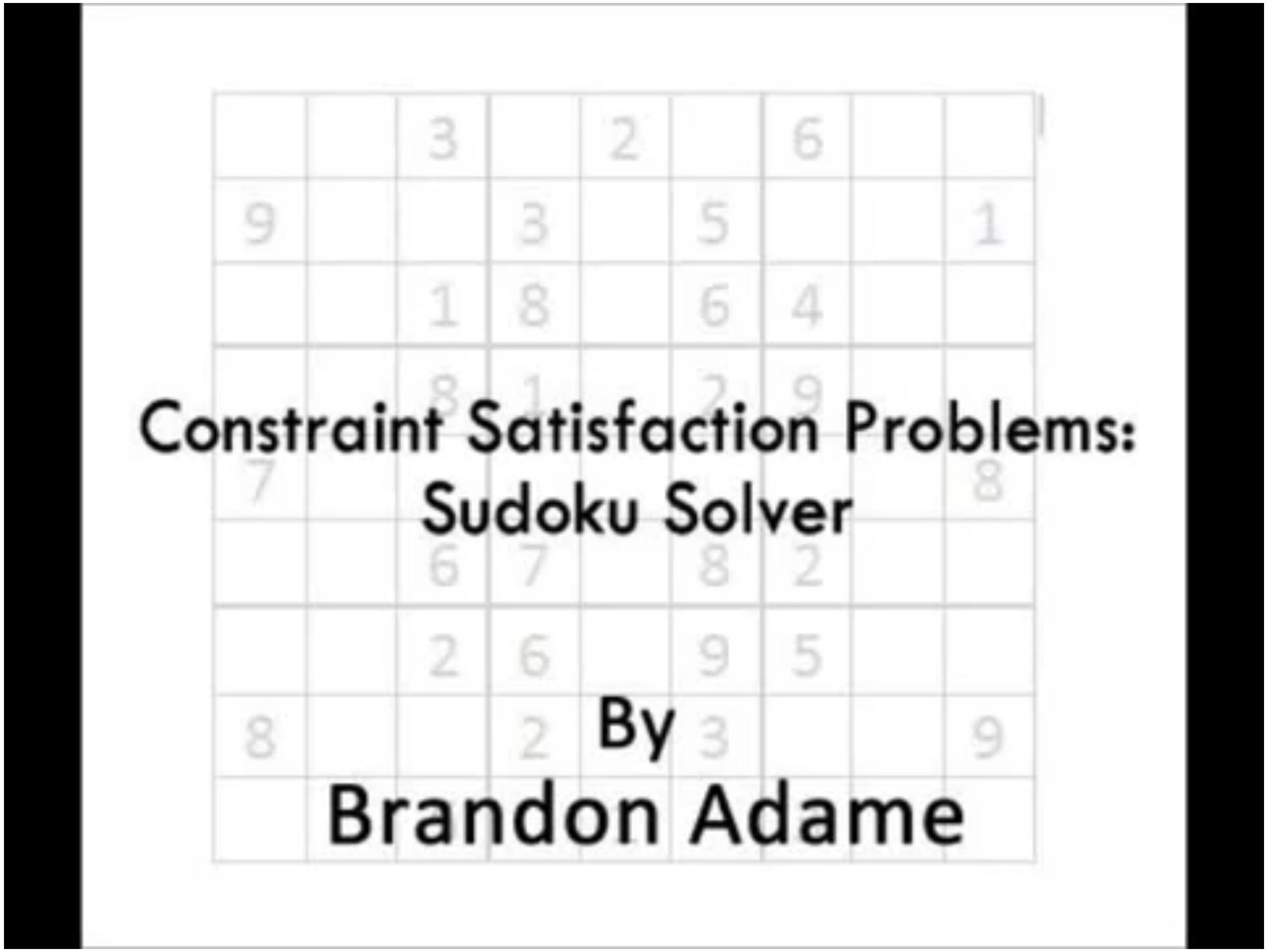
- Chronological Backtracking
  - Basic policy: when the search fails, back up to the preceding variable and try a different value, i.e. the most recent decision point is revisited.
- Alternative: go all the way back to one of the set of variables that caused the failure.
  - Conflict set
- Backjumping: backtracks to the most recent variable in the conflict set

Sudoku

	2		3
		4	2
3	4		
2		3	

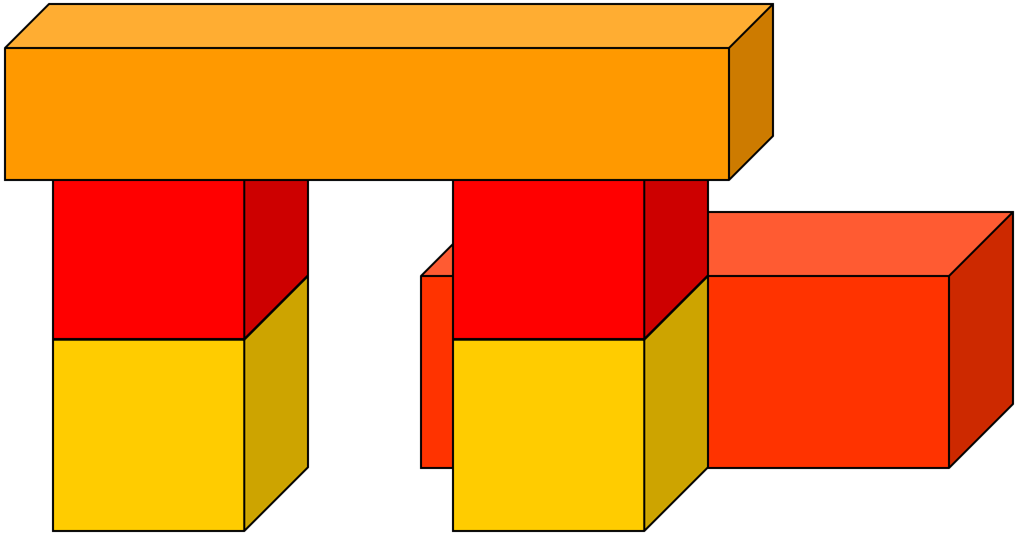
Solve the Puzzle

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

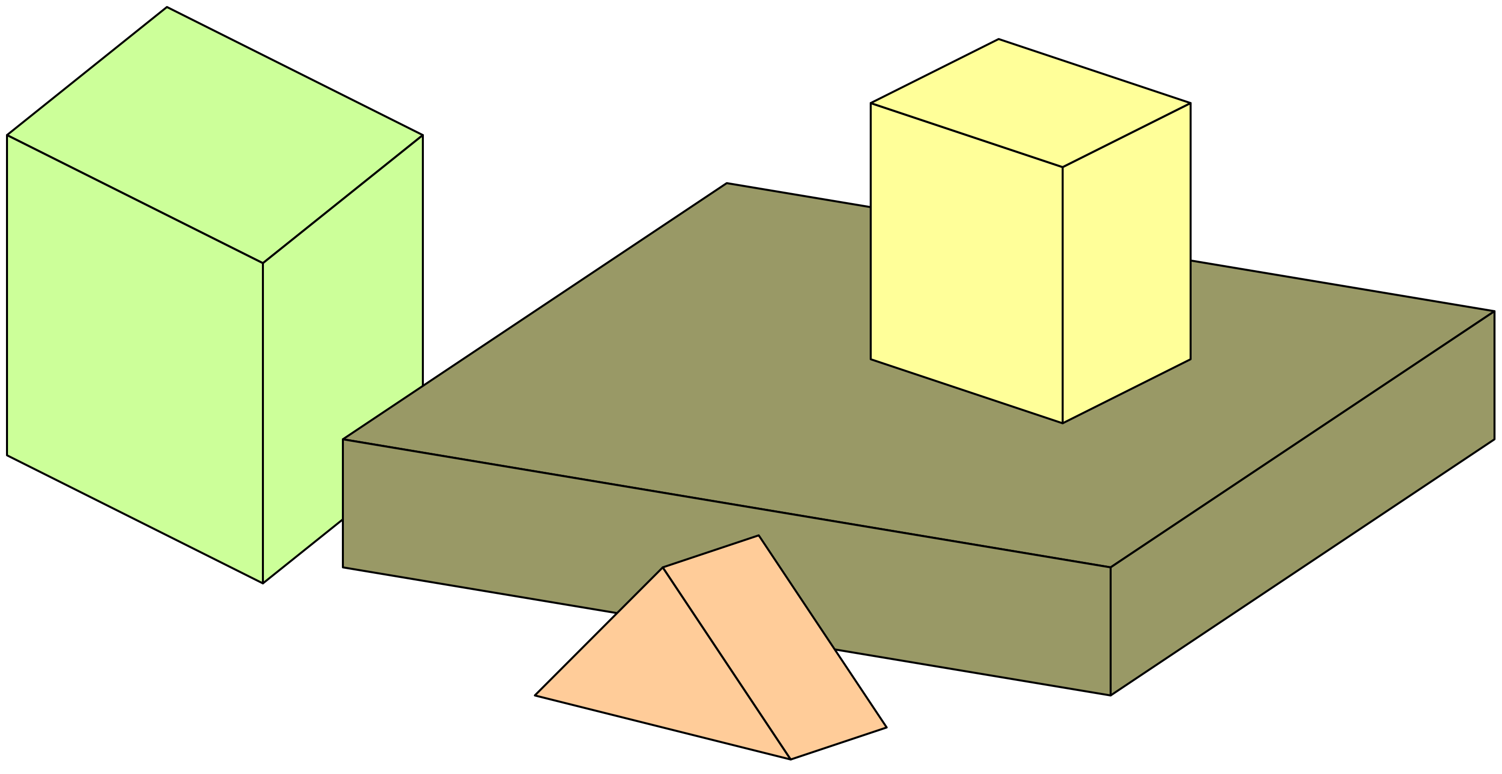

Constraint Satisfaction Problems:
Sudoku Solver
By
Brandon Adame

Waltz Labeling Algorithm

Example: Scene Analysis for Polyhedra



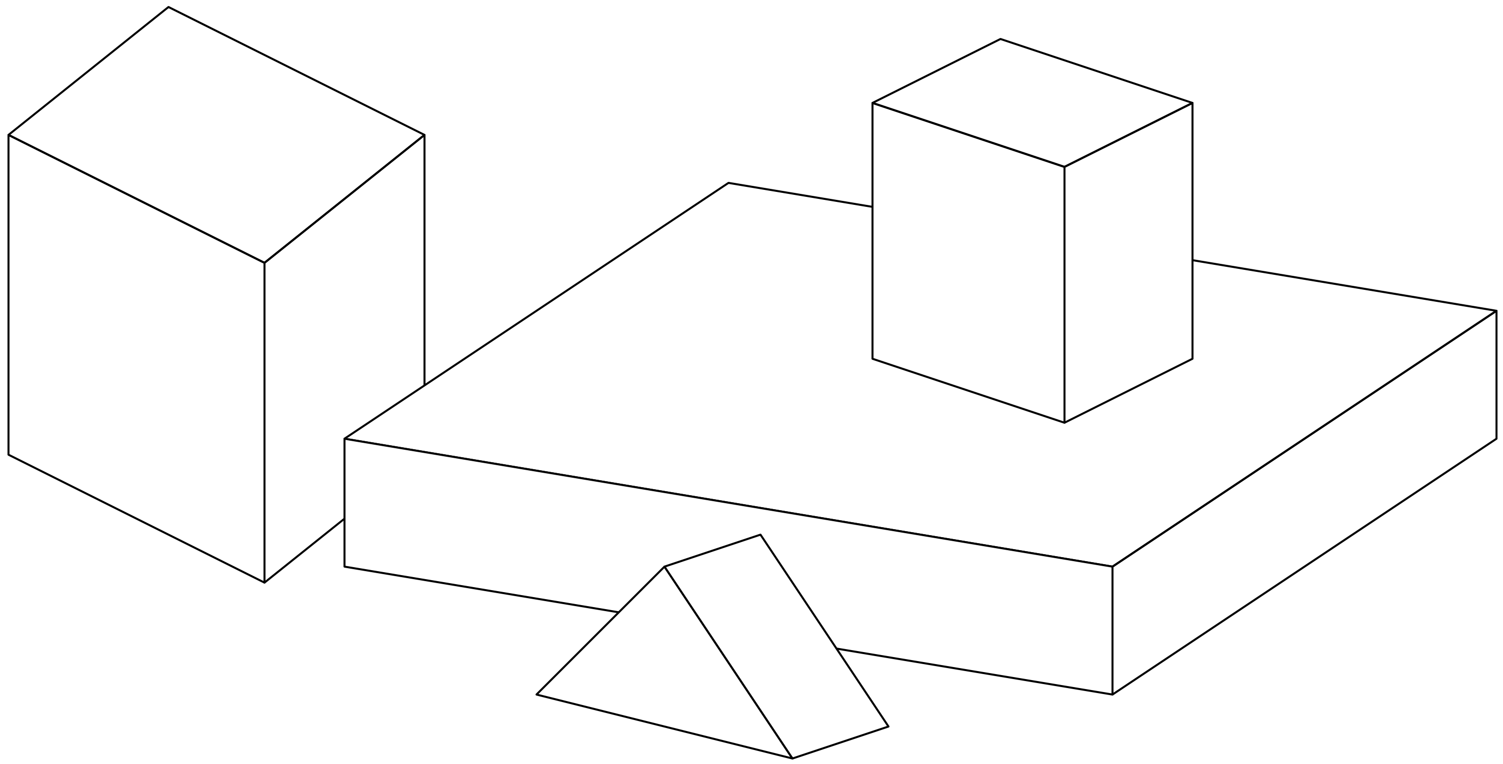
Edge Labeling



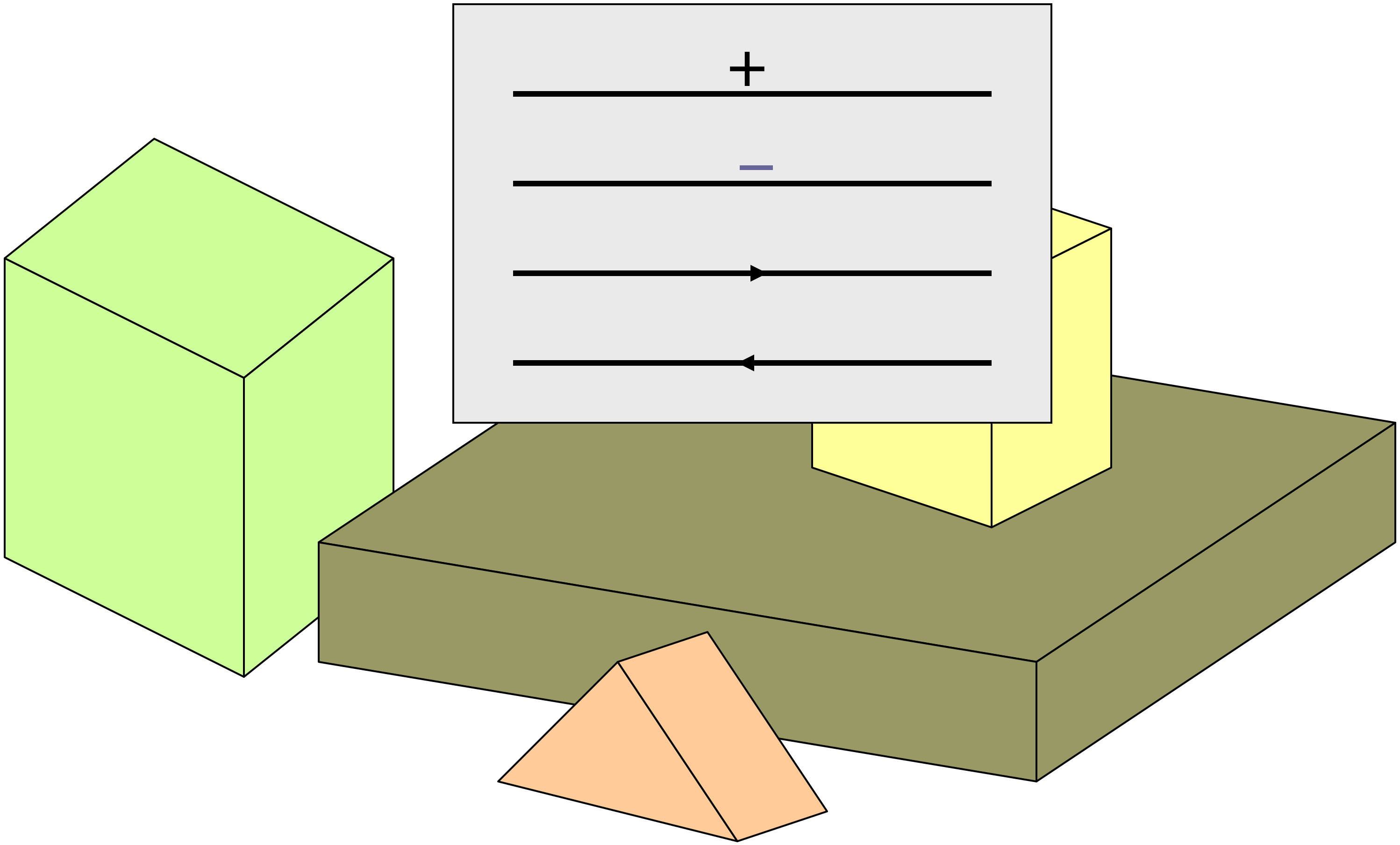
Edge Labeling as a CSP

- A variable is associated with each junction
- The domain of a variable is the label set of the corresponding junction
- Each constraint imposes that the values given to two adjacent junctions give the same label to the joining edge

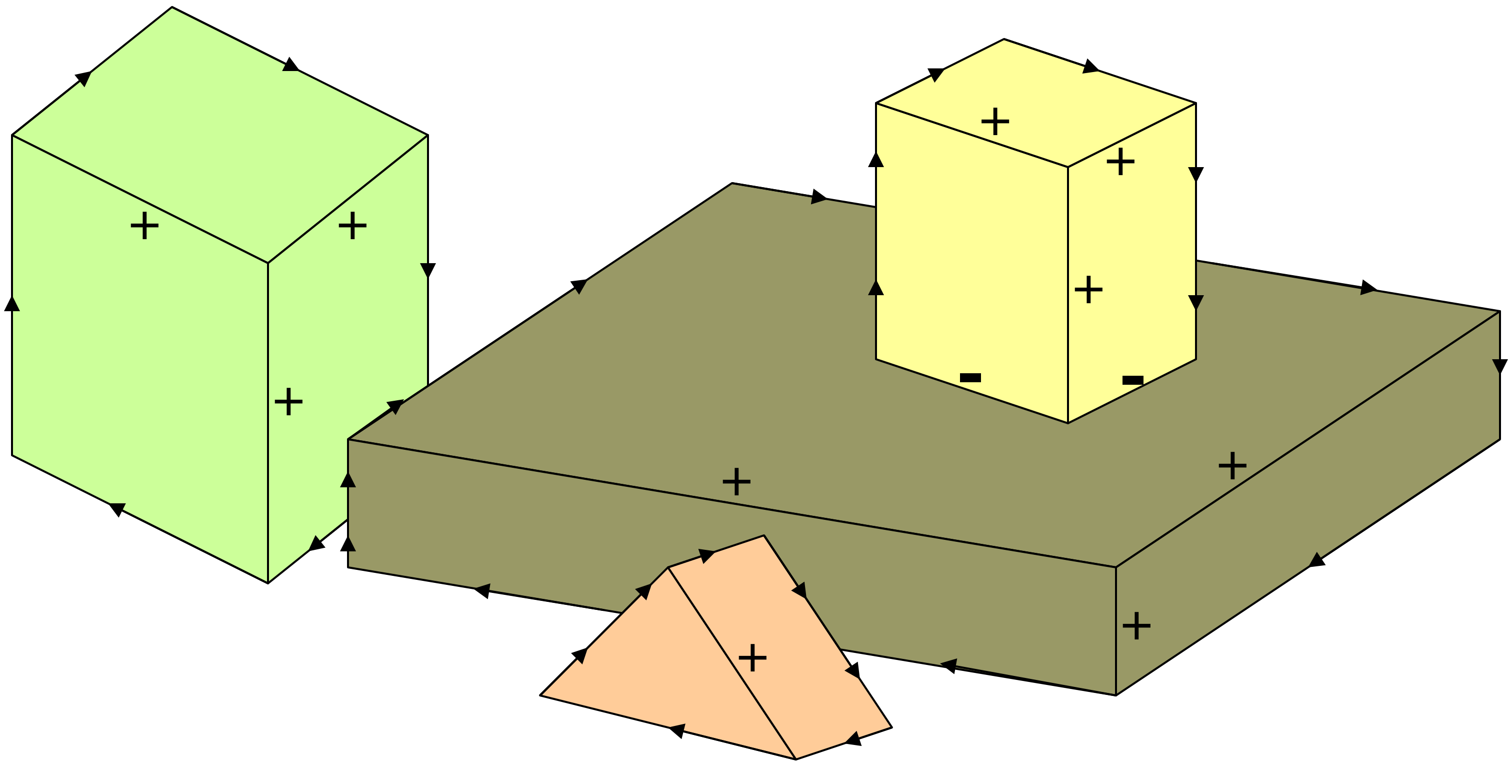
The Line Labeling Problem



Edge Labeling



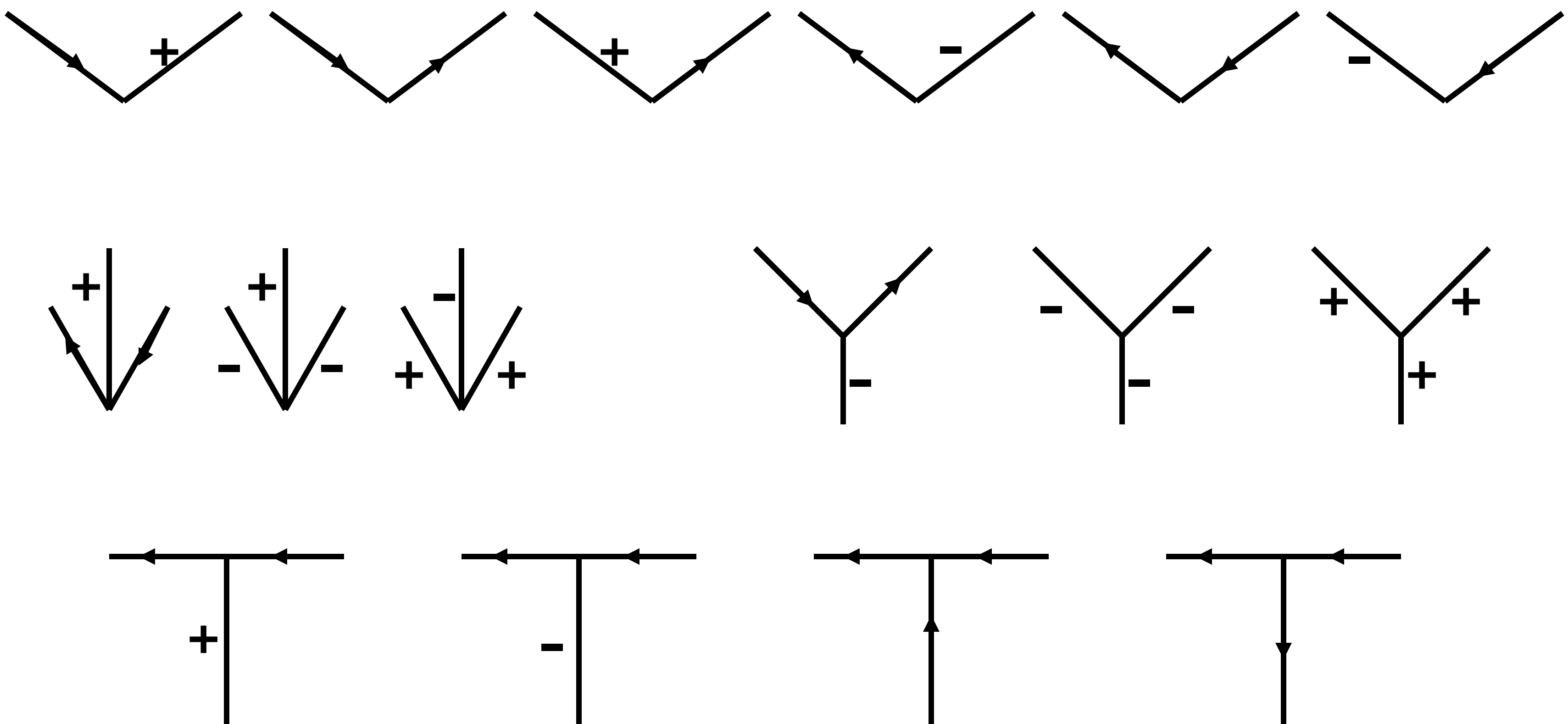
Edge Labeling



Combinatorics: Simple Case

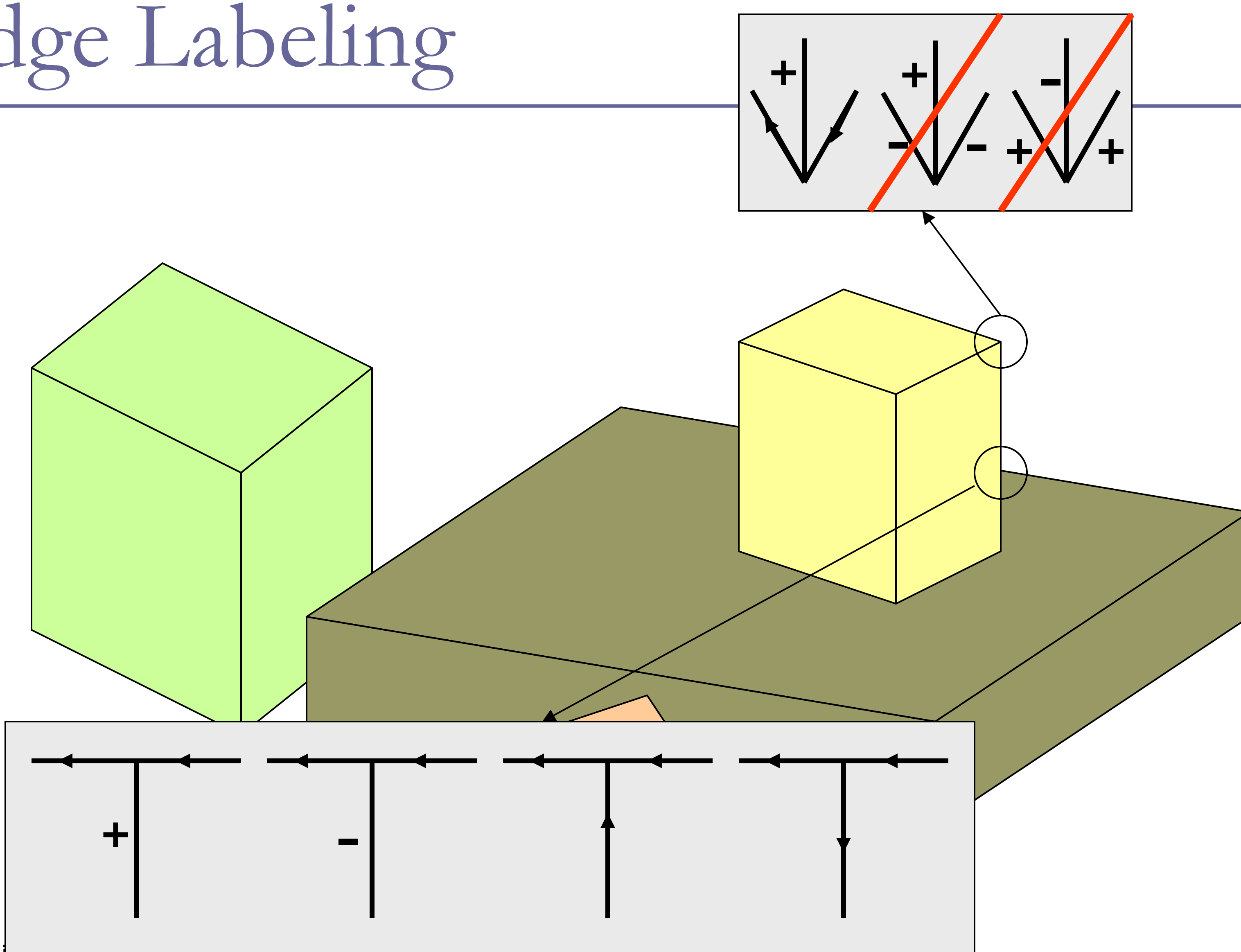
- Trihedral vertices, no concave or crack edges, uniform illumination (no shadows)
  - 4 ways to label a line, so
    - $4^2 = 16$ labelings for an L
    - $4^3 = 64$ labelings for fork, arrow, and T
- BUT out of 208 junctions, only 18 are physically possible.

The Huffman-Clowes Label Sets

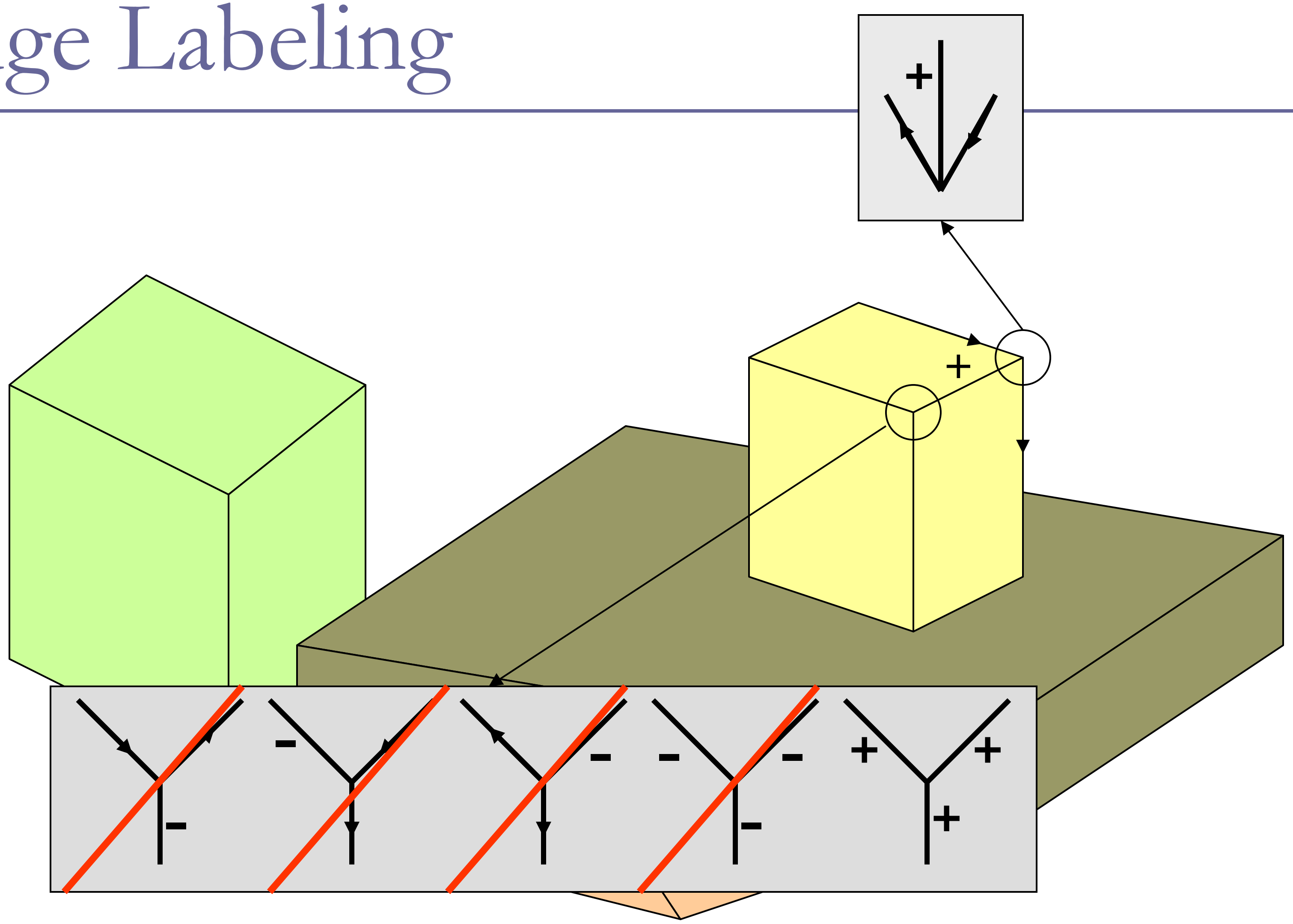


(Waltz, 1975; Mackworth, 1977)

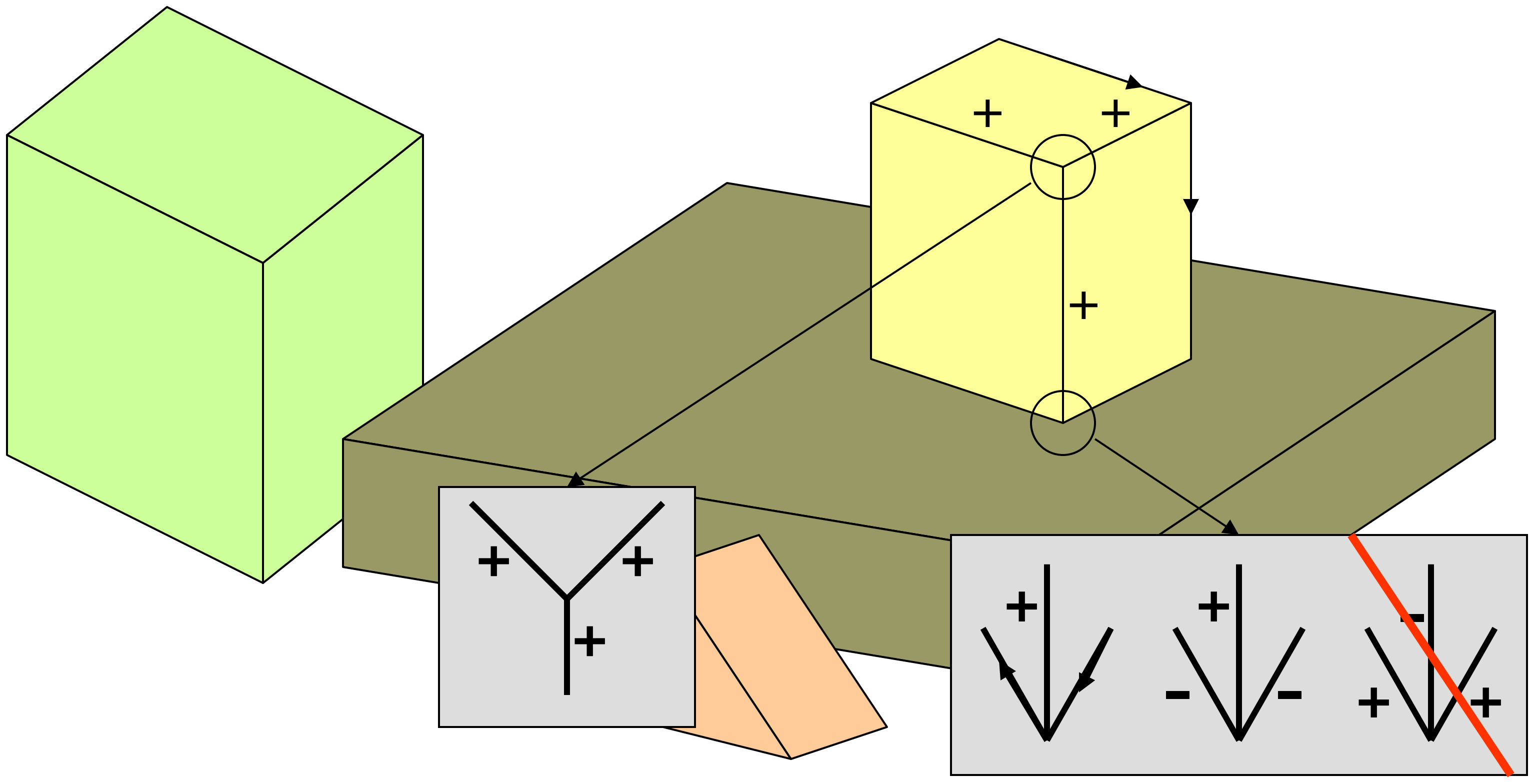
Edge Labeling



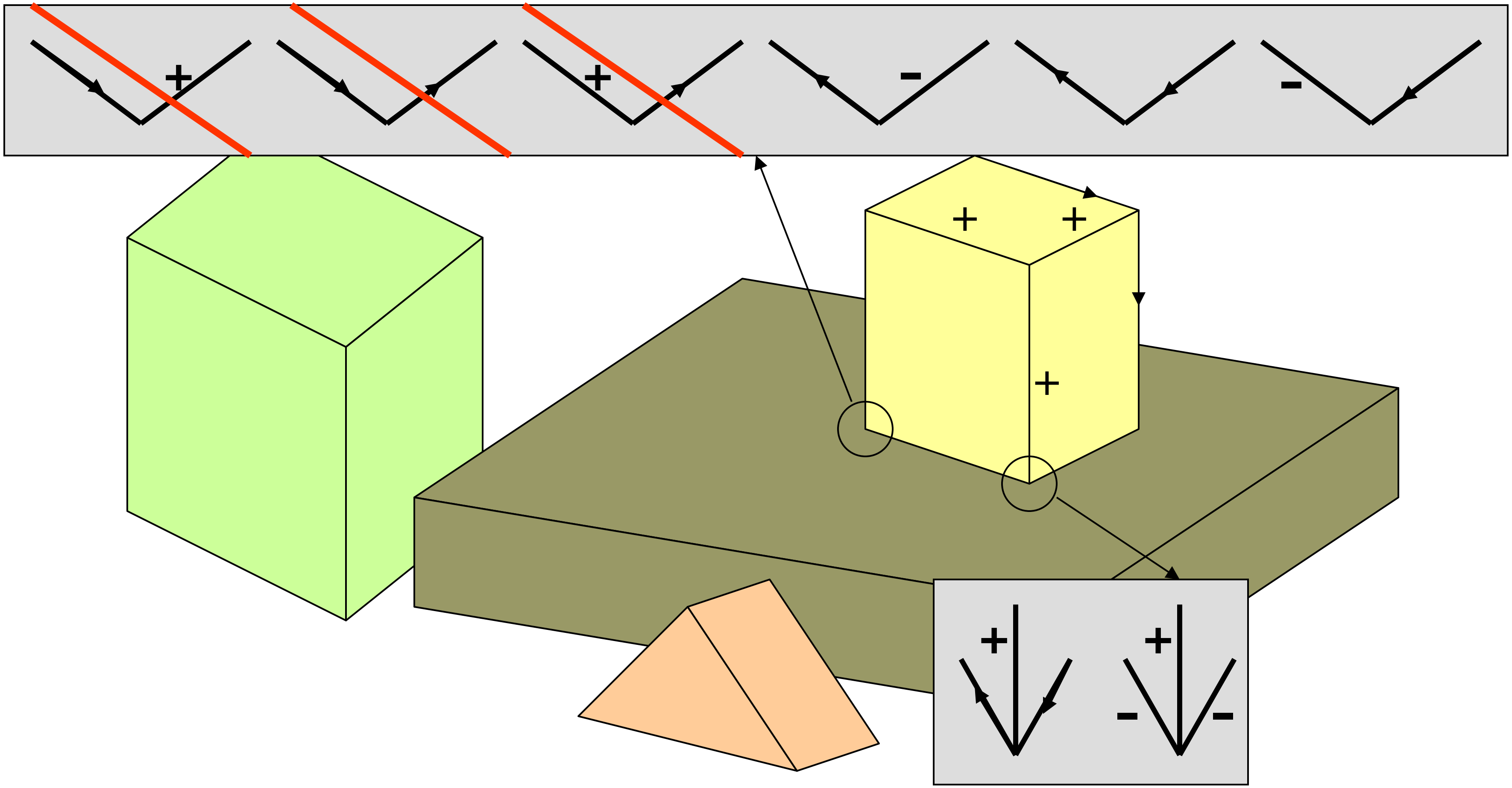
Edge Labeling



Edge Labeling



Edge Labeling



Waltz's Algorithm

1. Identify and label the lines at the border.
2. Label each vertex with all possible labelings for its vertex type.
3. Pick a vertex (starting from the border), *V*

For each neighboring vertex, *N*,

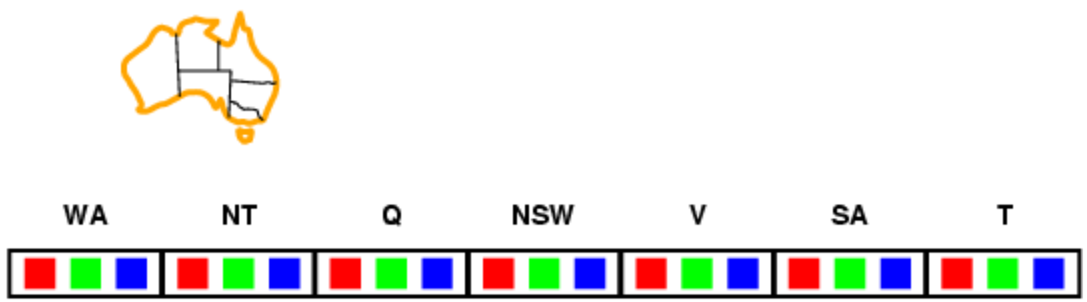
- If N and V agree on the possible labelings for the line between them, do nothing.
- Otherwise, remove the inconsistent labelings.
- Propagate the constraint by repeating the process for all neighboring vertices.

Termination condition:

- Every vertex has been visited at least once.
- There are no more constraints to propagate.

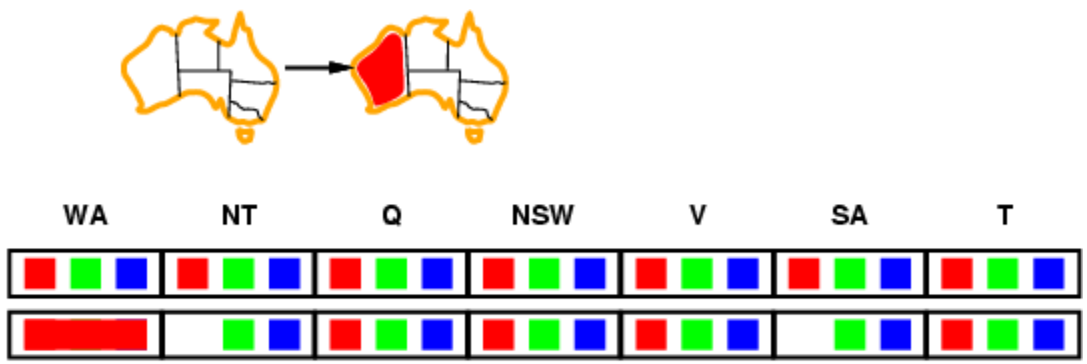
Forward Checking

- **Idea**:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -



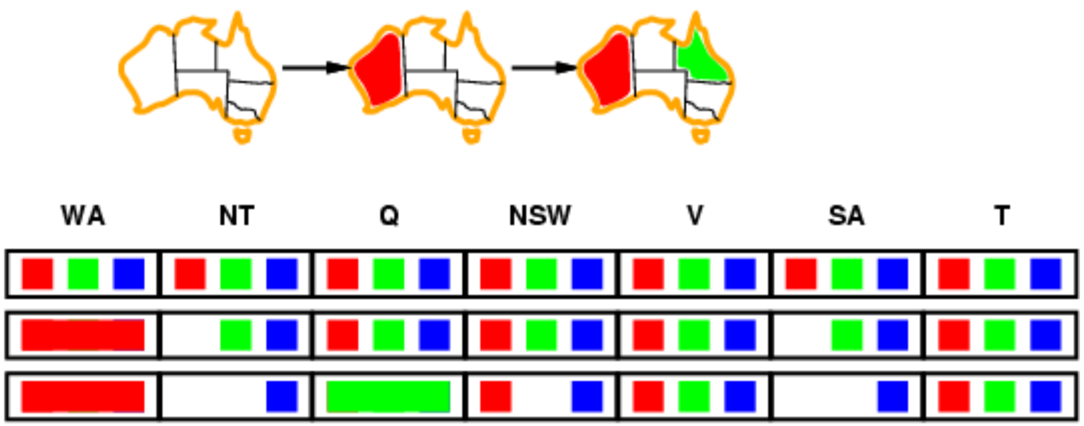
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -



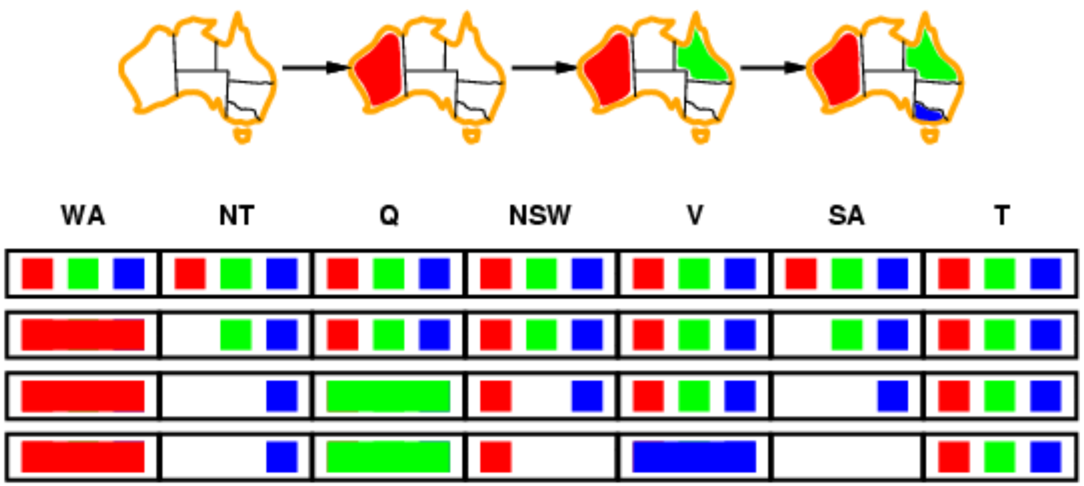
Forward Checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -

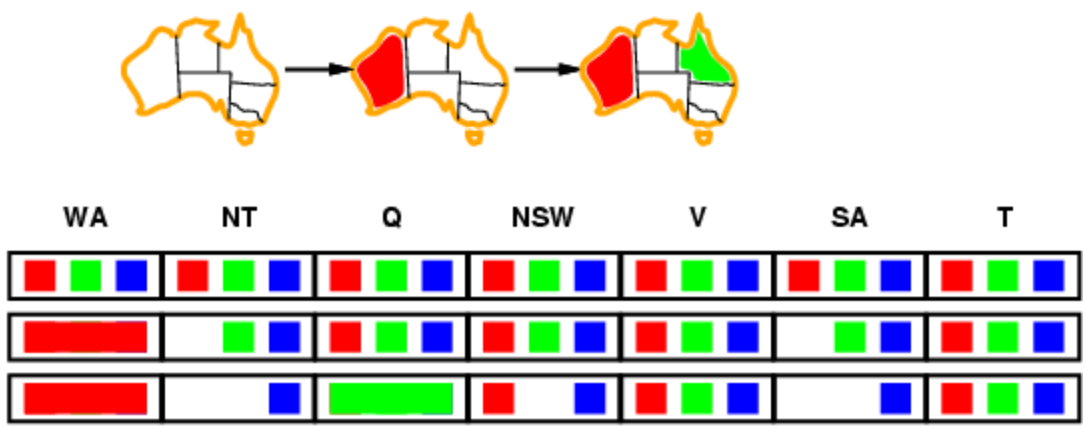


Forward Checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -



Constraint Propagation

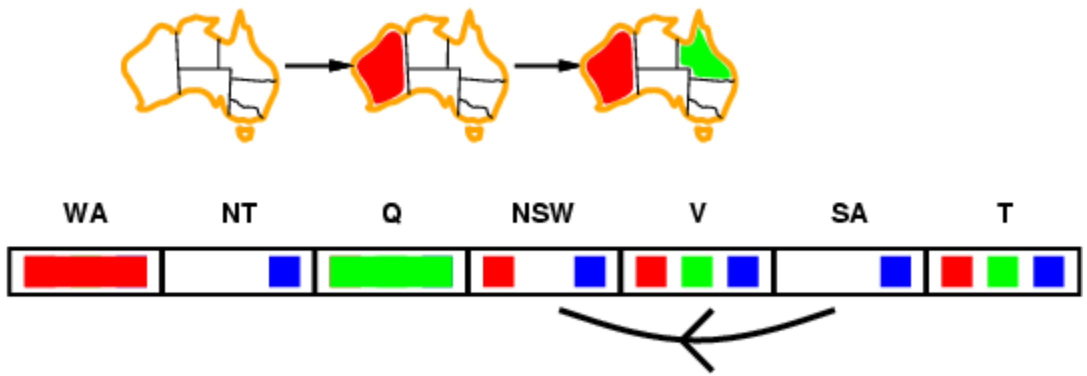
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:
- 

- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

Arc Consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

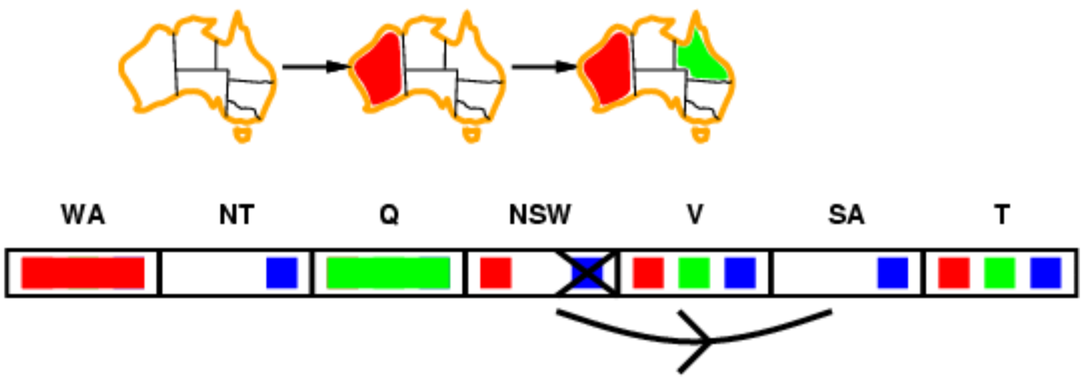
  for every value x of X there is some allowed y



Arc Consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

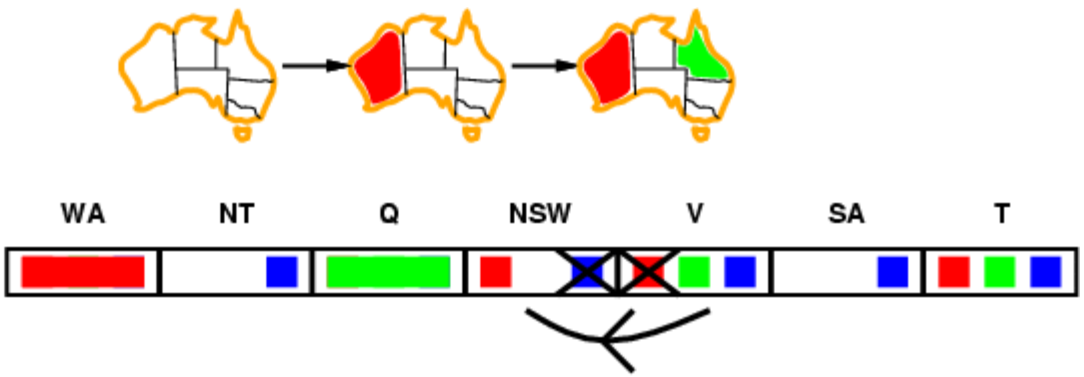
  for every value x of X there is some allowed y



Arc Consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

  for every value x of X there is some allowed y

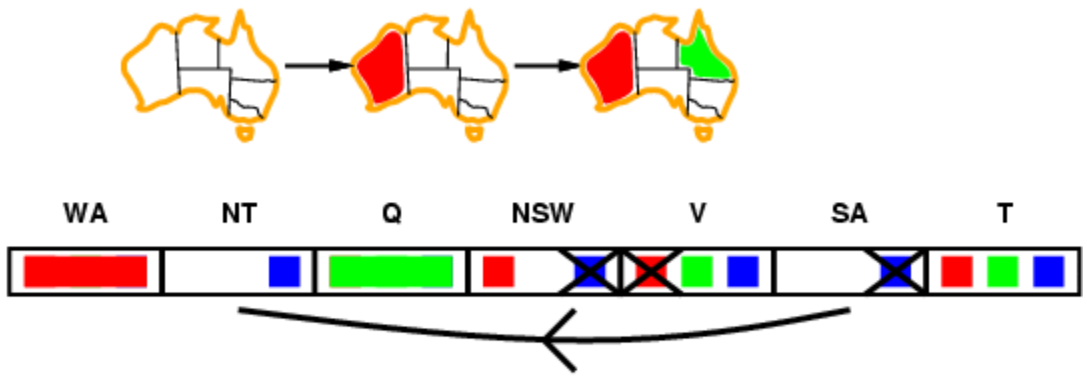


- If X loses a value, neighbors of X need to be rechecked

Arc Consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ 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

Arc Consistency Algorithm AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
        add (X_k, X_i) to *queue*

---

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint $X_i \leftrightarrow X_j$
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

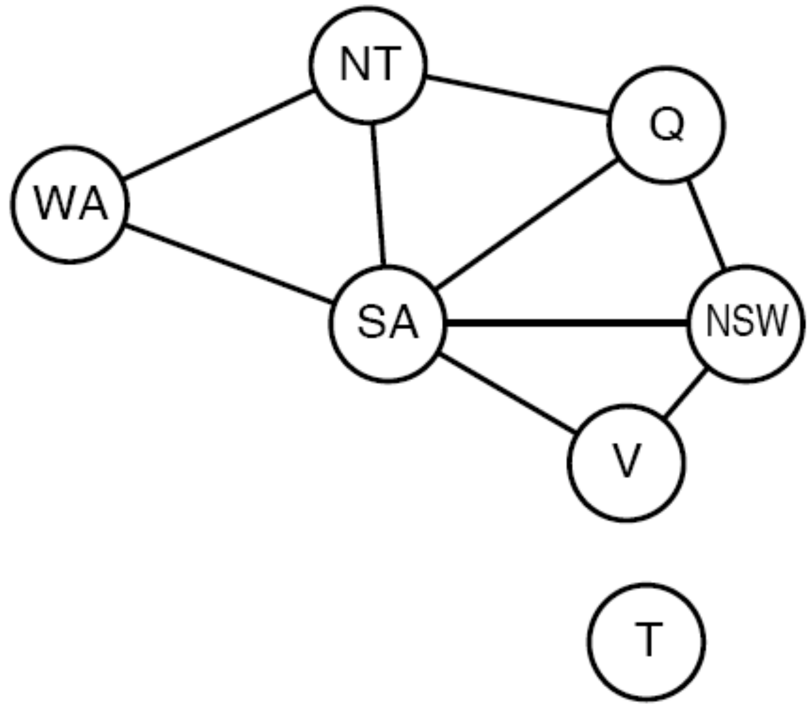
$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting **all** is NP-hard)

n: the number of variables; d: the size of domain

k-Consistency

- A CSP is *k*-consistent if, for any set of *k*-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to the *k*th variable.
  - Node-consistency
  - Path-consistency
- A graph is strongly *k*-consistent if it is *k*-consistent and is also (*k*-1)-consistent, (*k*-2)-consistent,…all the way to 1-consistent.

Problem Structure



Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph

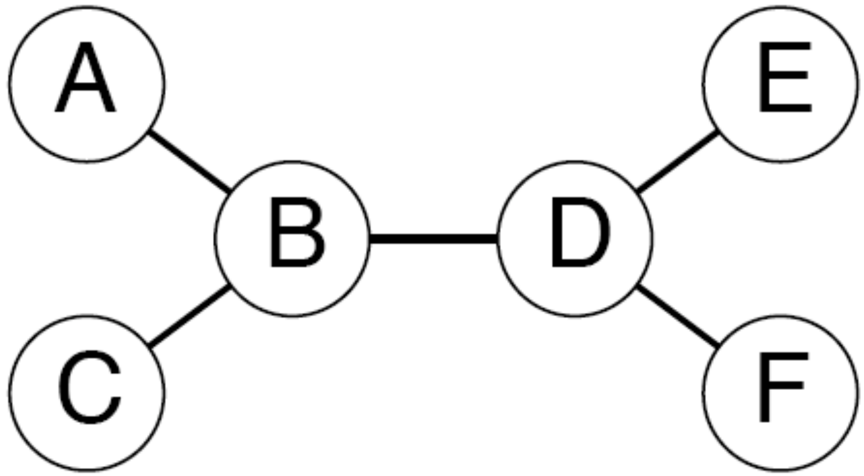
Analysis

- Suppose each subproblem has *c* variables out of *n* total
- Worst-case solution cost is linear in *n*.

$$n/c \bullet d^c$$

- Example: *n=80, d=2, c=20*
  - 2^{80}= 4 billion years at 10 million nodes/sec
  - 4 x 2^{20} = 0.4 seconds at 10 million nodes/sec

Tree-Structured CSPs

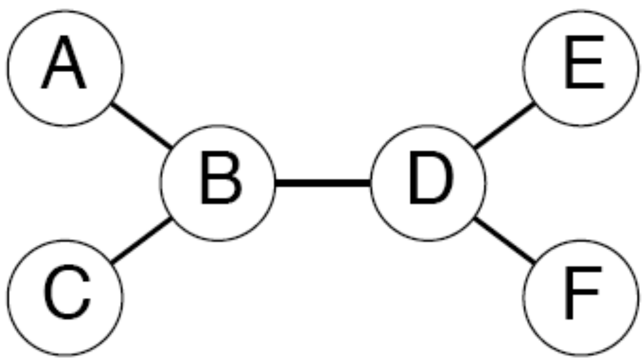


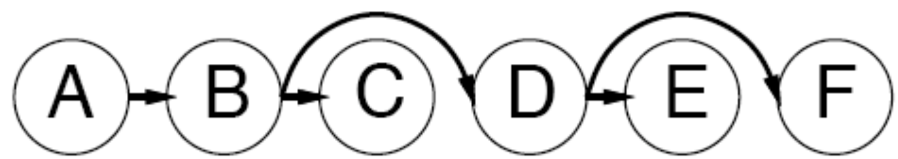
Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time

Compare to general CSPs, where worst-case time is $O(d^n)$

Algorithm

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering





2. For j from n down to 2, apply $\textsc{RemoveInconsistent}(Parent(X_j), X_j)$

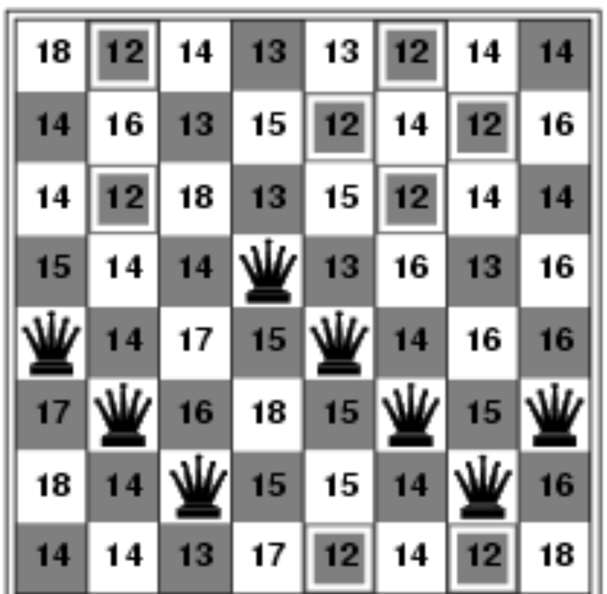
3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

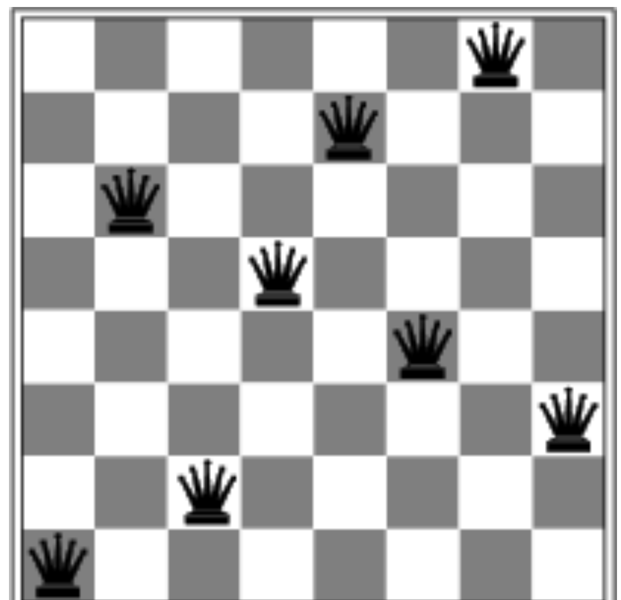
Local Search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with $h(n)$ = total number of violated constraints

Evaluation Function

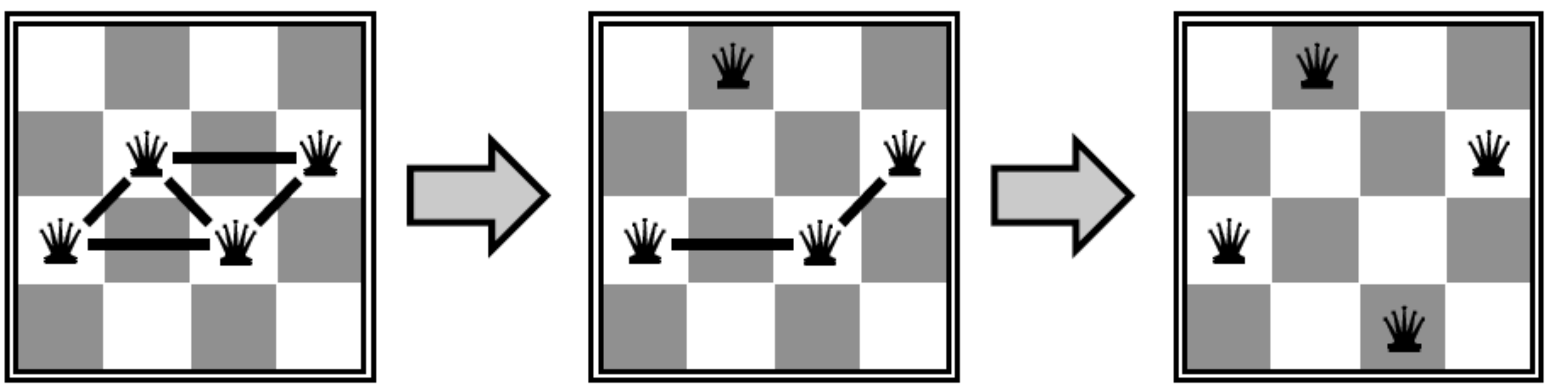
- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
  - *h* = 17 for the state on the left
  - *h* = 1 for the state (local minimum) on the right





Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
  - States: any configuration of n queens
    - E.g. 4 queens in 4 columns (4^4 = 256 states)
  - Actions: move any queen within its column
  - Goal test: no attacks
  - Evaluation: $h(n)$ = number of attacks



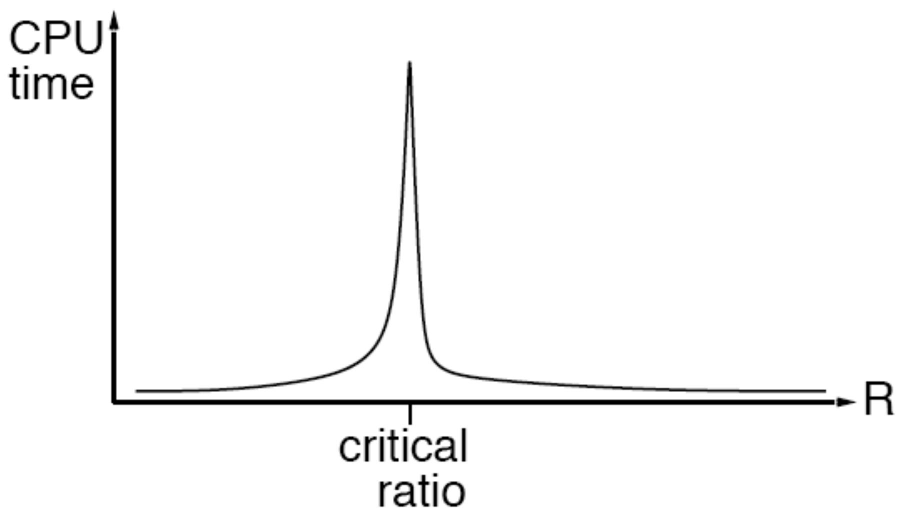
h = 5 **h = 2** **h = 0**

Performance of min-conflicts

Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10{,}000{,}000$)

The same appears to be true for any randomly-generated CSP **except** in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Iterative min-conflicts is usually effective in practice