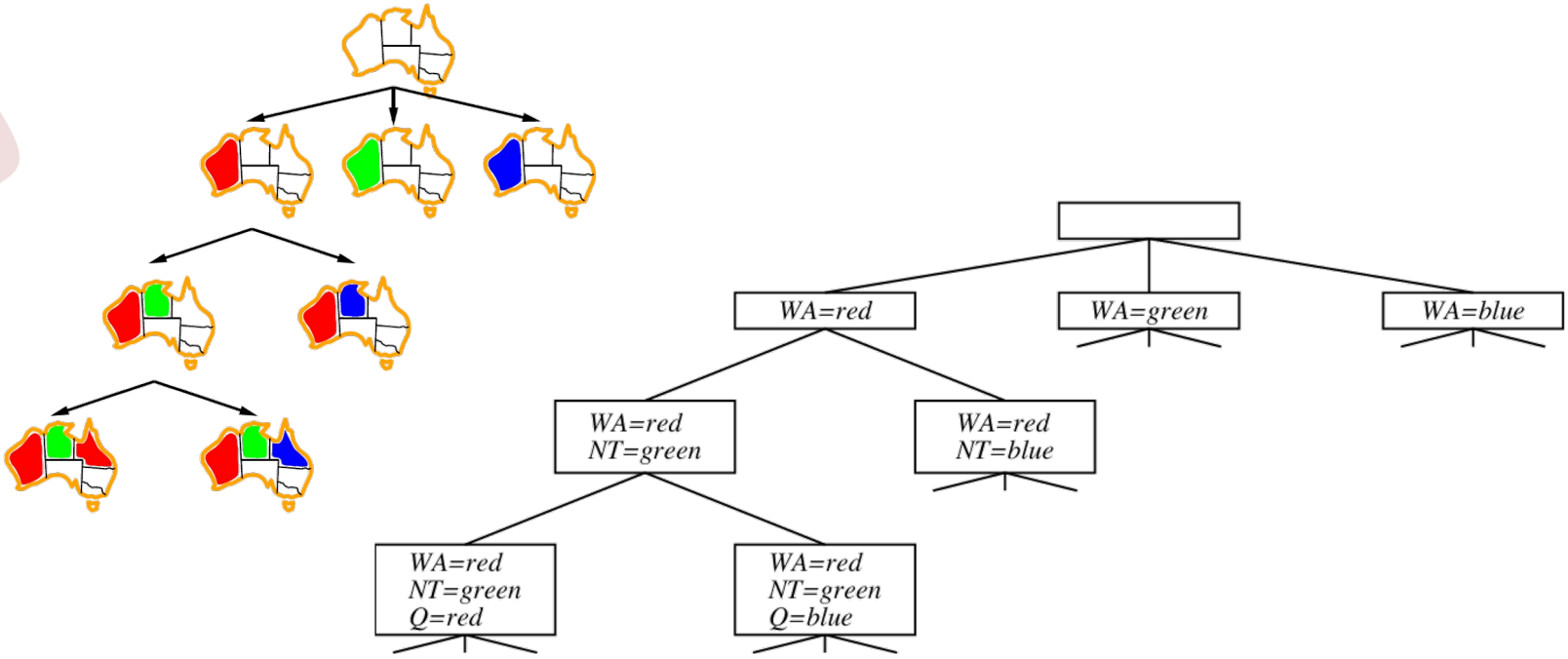
AI101

Lecture 5: Constraint Satisfaction Problems



Recap
Local Search and Adversarial Search

If we do not care about the path to a solution, iterative local improvements could help:

- Hill Climbing
- Gradient Descent
- Simulated Annealing

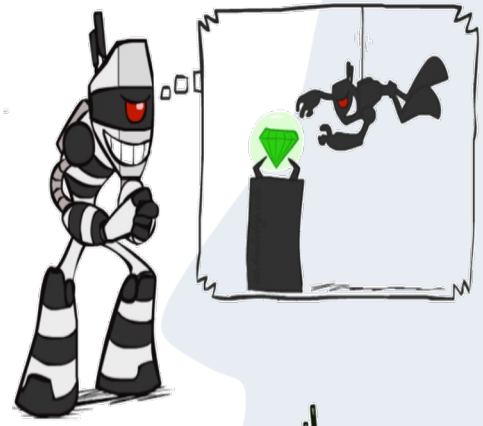
But we might have an opponent. Then we need techniques of adversarial search

- Minimax
- Alpha-Beta Pruning

Today: Constraint Satisfaction Problems

Why?

Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space



Planning: sequences of actions

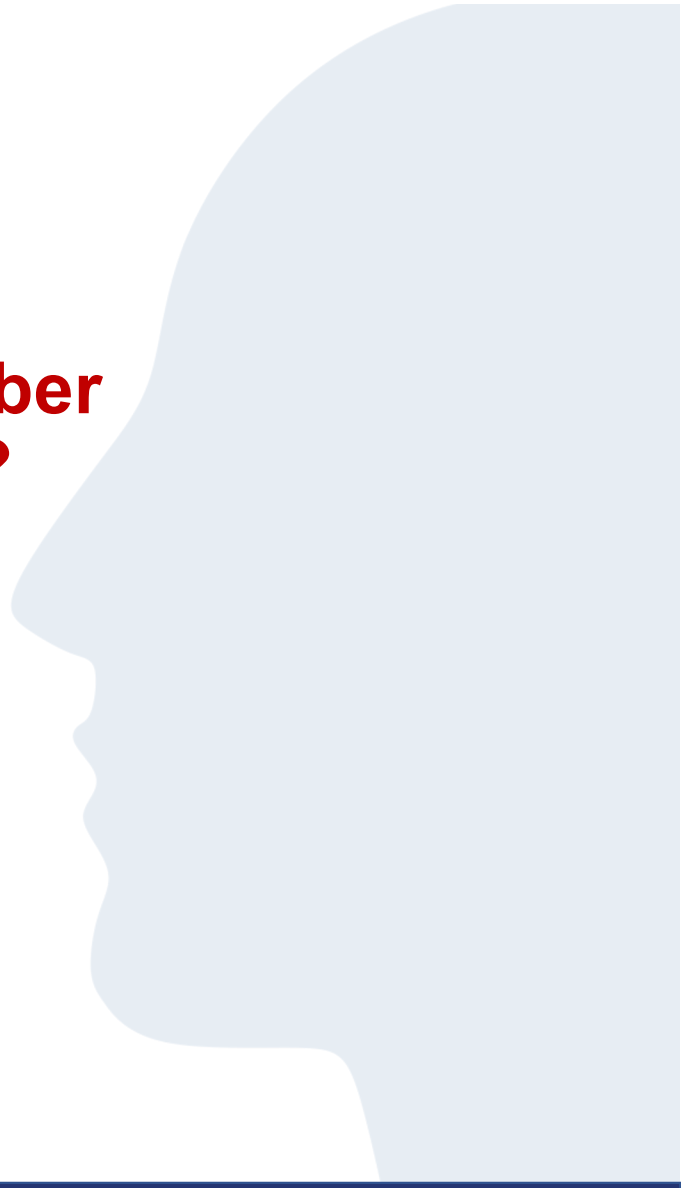
- The path to the goal is the important thing
- Paths have various costs, depths
- Heuristics give problem-specific guidance

Identification: assignments to variables

- The goal itself is important, not the path
- All paths at the same depth (for some formulation
- CSPs are specialized for identification problems

Chris Amato, Northeastern University resp. Rob Platt, CS188 UC Berkeley, AIMA

Today: Constraint Satisfaction Problems
Outline



What if solutions are described by a number of constraints that the state must satisfy?

- Backtracking Search
- Forward Checking
- Constraint Propagation
- Local Search
- Tree-Structured CSPs

Constraint Satisfaction Problems

Constraint Satisfaction Problem

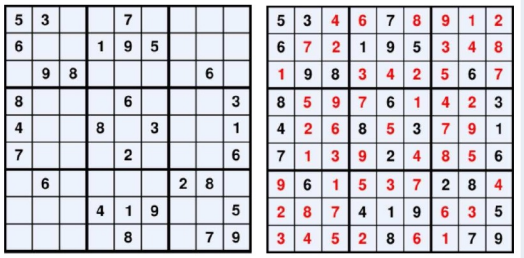
Constraint satisfaction is a technique where a problem is solved when its solution satisfy certain constraints or rules of the problem.

Components are

- A State, defined by variables X_i with d values from domain D_i
- A Goal test, defined as a set of constraints C specifying allowable combinations of values for subsets of variables

Solving Constraint Satisfaction Problems requires

- A state-space
- The notion of the solution



Sudoko

cryptoarithmetic puzzle

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

Constraint Satisfaction Problems
Real World CSPs

Assignment problems

- e.g., who teaches what class

Timetabling problems

- e.g., which class is offered when and where?

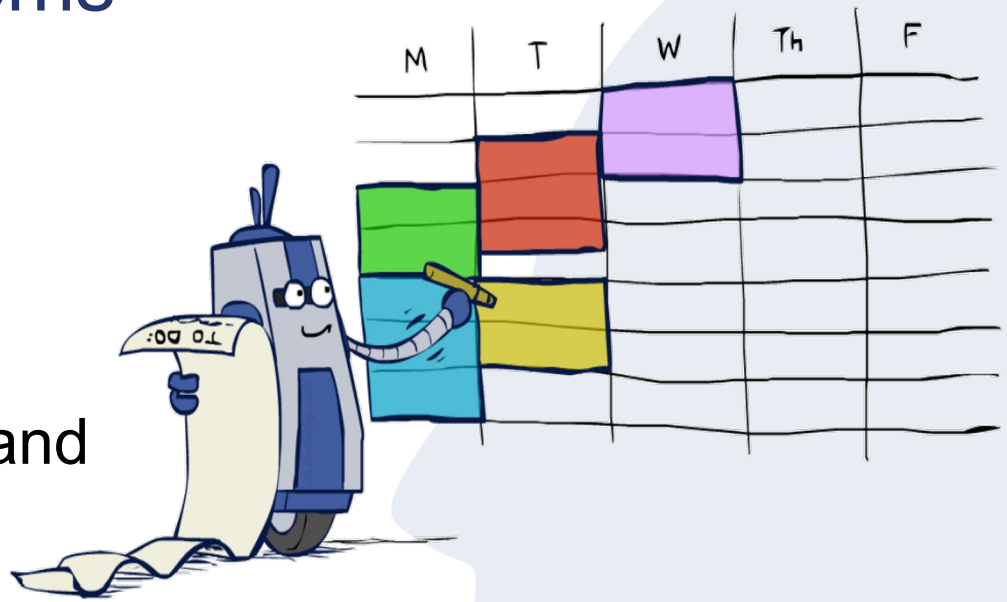
Hardware configuration

Spreadsheets

Scheduling

Floor planning

…



Note that many real world problems involve real-valued variables

- Linear constraints solvable in polynomial time using linear programming
- But problems with nonlinear constraints are undecidable

Constraint Satisfaction Problems
Assignment of values to variables

A state in state-space is not a blackbox anymore (as in standard search) but defined by assigning values to some or all variables such as

$$X_1 = v_1, X_2 = v_2, \ldots$$

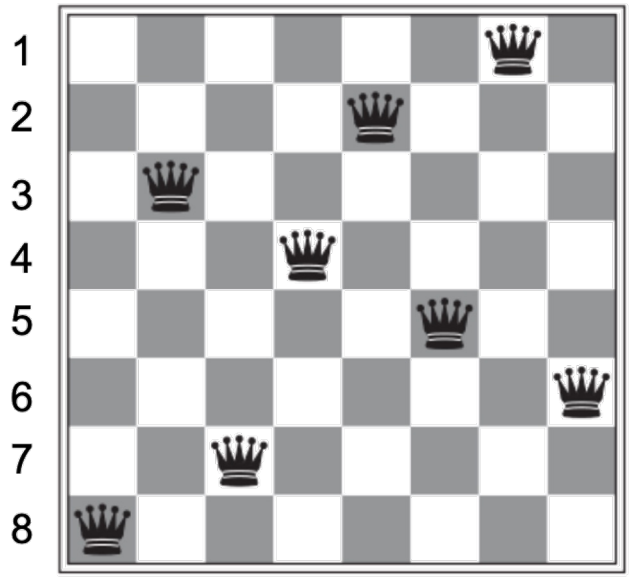
Can be done in three ways:

1. **Consistent/Legal Assignment**
   An assignment which does not violate any constraint or rules
2. **Complete Assignment**
   An assignment where every variable is assigned with a value, and the solution to the CSP remains consistent.
3. **Partial Assignment**
   An assignment which assigns values to some of the variables only.

Since we do not care about the path to a solution and solutions are described by constraints, we can do better than search trees ((un)informed search) but still maintain a data structure (compared to local search)

Constraint Satisfaction Problems

Example: n-Queens



Problem: place *n* queens on an *nxn* chessboard such that no two queens threaten each other

Variables: $X =$ One variable for each row (i.e, each queen)

Domain of variables: $D =$ A number between 1 and 8

Constraints: $C =$ Enumeration of disallowed configurations

Chris Amato, Northeastern University resp. Rob Platt, CS188 UC Berkeley, AIMA

Constraint Satisfaction Problems
Constraint Graphs

Why do we want to build a (constraint) graph

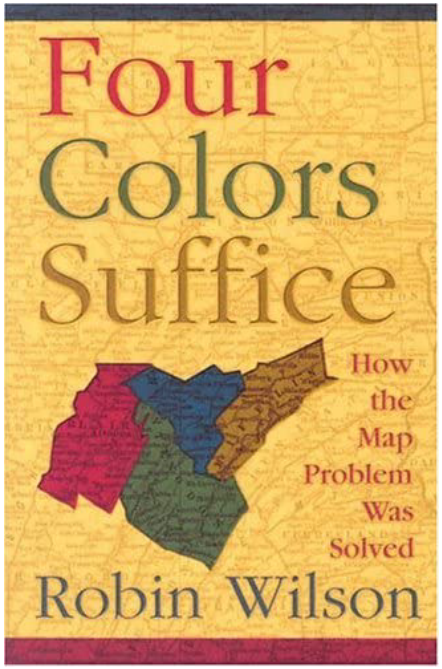
- Abstraction of the problem makes it easier to solve
- Abstraction often makes it easier to understand the problem

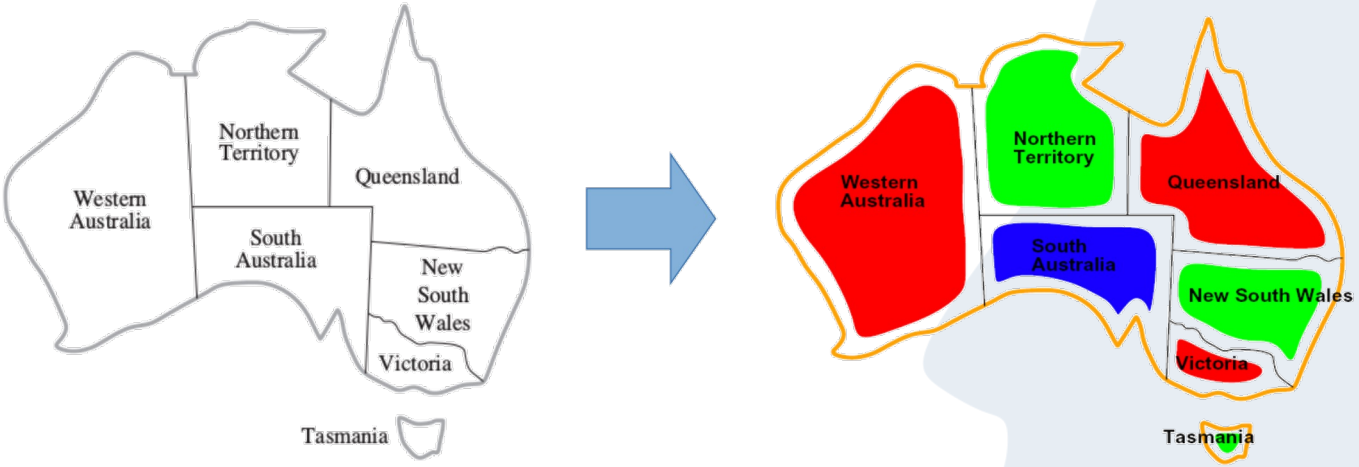
How do we build a constraint graph

- Every variable is represented as a node
- Every edge indicates a constraint between them

Since we do not care about the path to a solution and solutions are described by constraints, we can do better than search trees ((un)informed search) but still maintain a data structure (compared to local search)

Constraint Satisfaction Problems
Example: The Four Color Map Theorem





Problem: assign each territory a color such that no two adjacent territories have the same color

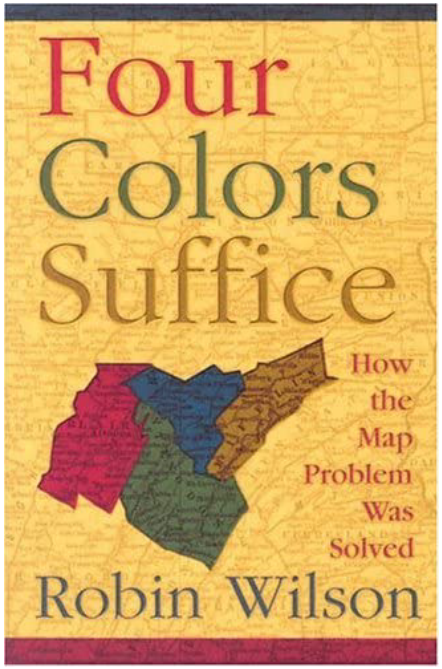
Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$

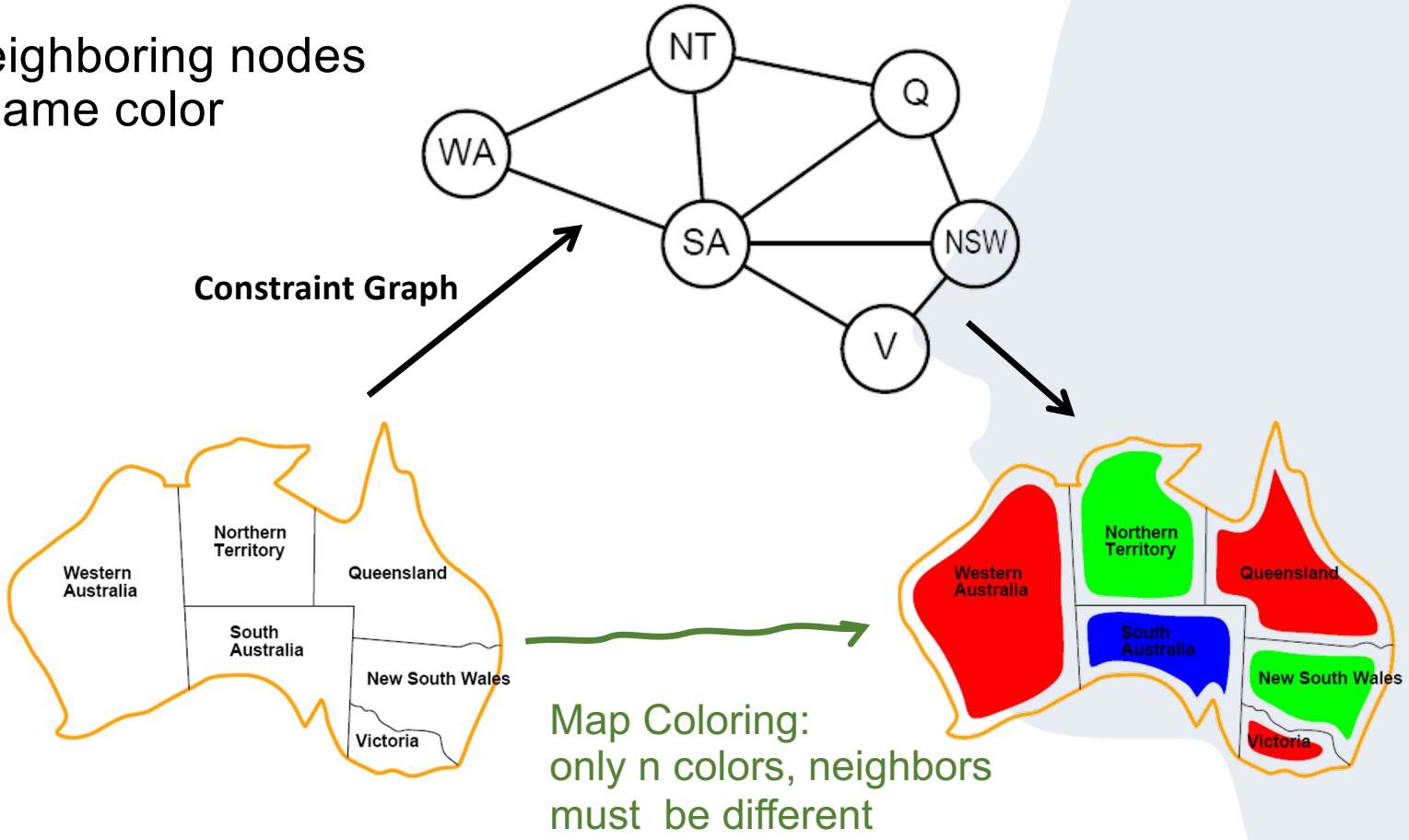
Domain of variables: $D = \{r, g, b\}$

Constraints: $C = \{SA \neq WA, SA \neq NT, SA \neq Q, \dots\}$

Constraint Satisfaction Problems
Example: The Four Color Map Theorem

Constraint: Two neighboring nodes must not have the same color





Constraint Satisfaction Problems
Example: The Four Color Map Theorem

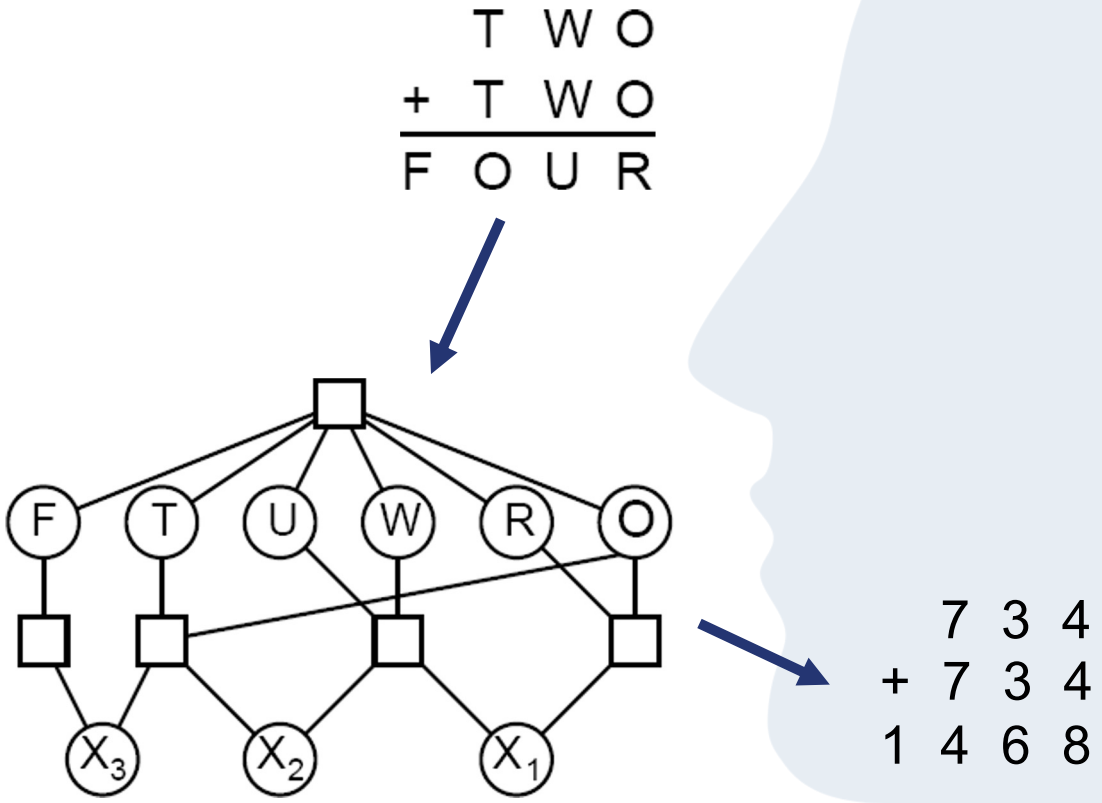
https://www.youtube.com/watch?v=NgbK43jB4rQ&t=513s

Constraint Satisfaction Problems
Example: TWO + TWO = FOUR

Constraints:

Connected nodes are involved in (in-)equations:

$$2 \cdot O = 10 \cdot X_1 + R$$
$$2 \cdot W + X_1 = 10 \cdot X_2 + U$$
$$2 \cdot T + X_2 = 10 \cdot X_3 + O$$
$$F = X_3$$
$$F \neq T \neq U \neq W \neq R \neq O$$



Constraint Satisfaction Problems
Example: Wildlife Corridor Design

https://www.youtube.com/watch?v=vPluRAznFPw&t=211s

Types of Constraints

Unary constraints involve a single variable,

- e.g., South Australia ≠ green

Binary constraints involve pairs of variables,

- e.g., South Australia ≠ Western Australia

Higher-order constraints involve 3 or more variables

- e.g., $2 \cdot W + X_1 = 10 \cdot X_2 + U$

Preferences (also called soft constraints)

- e.g., *red is better than green*
- are not binding, but task is to respect as many as possible

  → constrained optimization problems

How do we solve CSPs?

Two principal approaches:

1. Search:
   - successively assign values to variable
   - check all constraints
   - if a constraint is violated → backtrack
   - until all variables have assigned values
2. Constraint Propagation:
   - maintain a set of possible values D_i for each variable X_i
   - try to reduce the size of D_i by identifying values that violate some constraints

How do we solve CSPs?
Search

We map the CSPs into search problems:

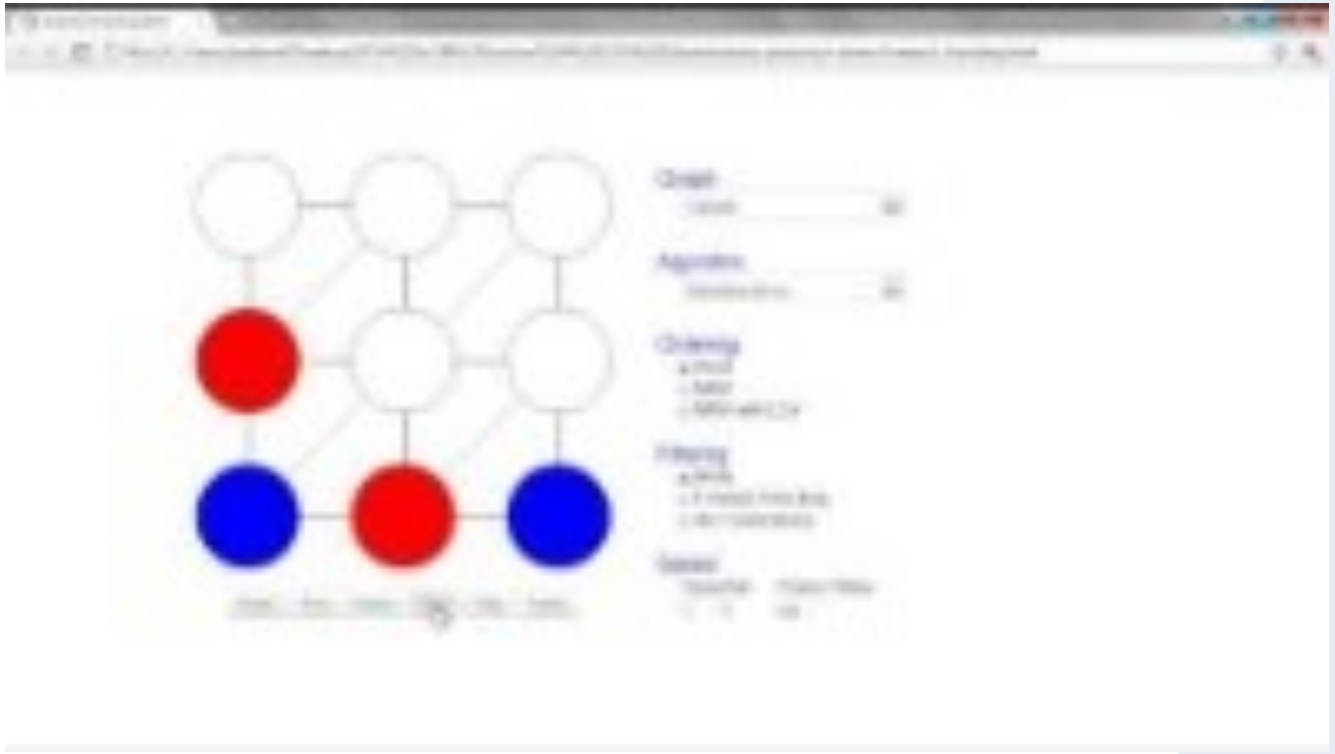
- Nodes = assignments of values to a subset of the variables
- Neighbors of a node = nodes in which values are assigned to one additional variable
- Start node = the empty assignment
- Goal node = a node which assigns a value to each variable and satisfies all constraints

How to build a search space, which is rather simple

- The start node is an empty assignment of values to variables
- Its successor are all possible ways of assigning a value to a variable (d=1)
- Their successors are those with two variables assigned (d=2)
- …

Which search strategy would you use?

How do we solve CSPs?
Naïve Search



https://youtu.be/Hv_JlWld9iQ?t=2048

How do we solve CSPs?
Complexity of Naïve Search

- **Assumptions**
  - we have n **variables**, so all all solutions are at depth n in the search tree
  - all variables have v possible **values**
- **Then**
  - at **level 1** we have $n \cdot v$ possible assignments, since we can choose one of n variables and one of v values for it
  - at **level 2**, we have $(n-1) \cdot v$ possible assignments for each previously unassigned variable, since we can choose one of the remaining $n-1$ variables and one of the v values for it
  - In general: branching factor at depth l: $(n-l+1) \cdot v$
  - $n \cdot (n\text{-}1) \cdot v \cdot v$ leaves in total at $d=2$

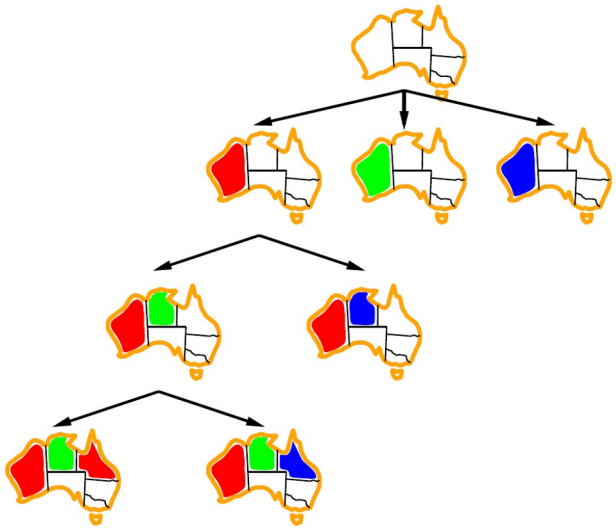
  …
- **Therefore**, the search tree has $\boldsymbol{n! v^n}$ leaves

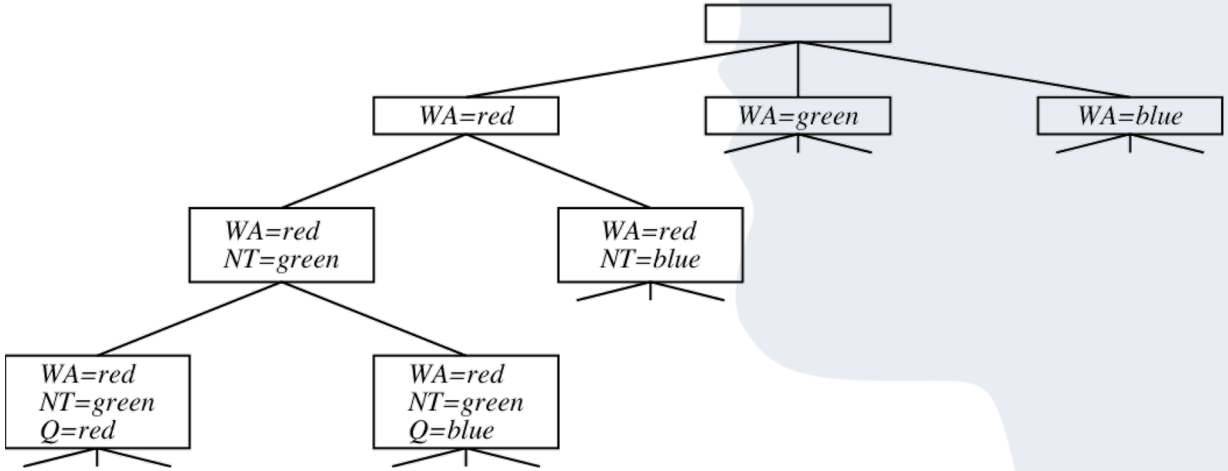
How do we solve CSPs?

Wec an do better: Commutative Variable Assignments

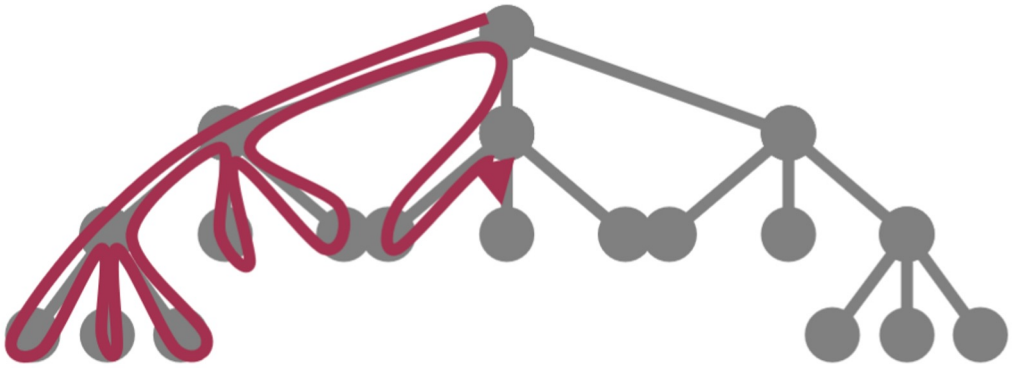
Is it relevent if we do [*WA = red* then *NT = green*] or [*NT = green* then *WA = red*]?

- No, assignments are commutative, order is not important
- Thus, at each node, we only need to make assignments for one of the variables
- Reduces the total complexity to *v*^*n*





Backtracking Search

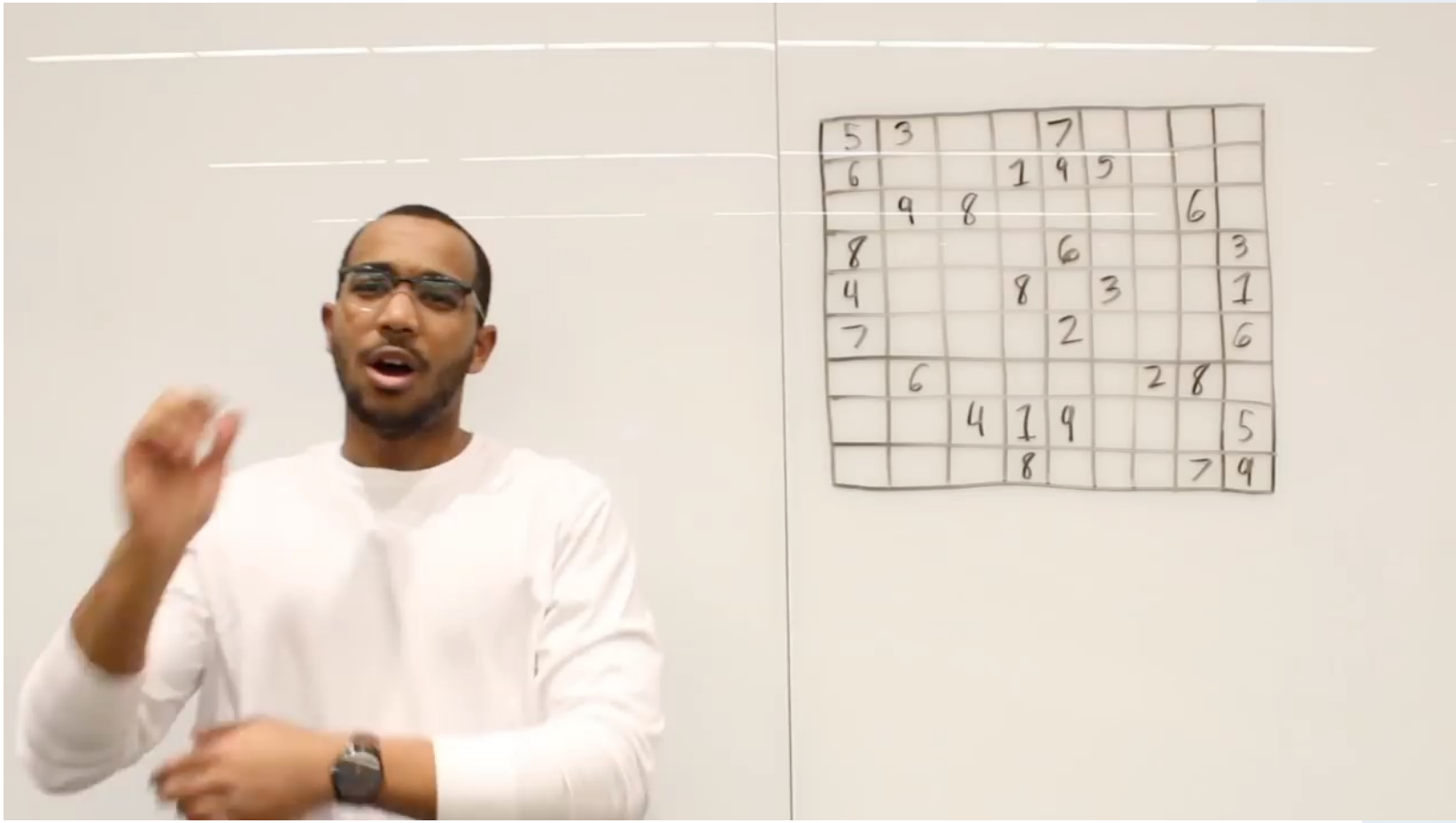


```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

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
```

- DFS with single variable assignments per level is also called **backtracking search**
- **Backtracking is the basic uninformed search algorithm for CSPs**
  - Add one constraint at a time without a conflict, succeed if legal assignment is found
  - Can solve the n-queens problem for up to *n=25*
- Complexity in worst case still exponential
- Heuristics for selecting and ordering variables can improve performance

Backtracking Search

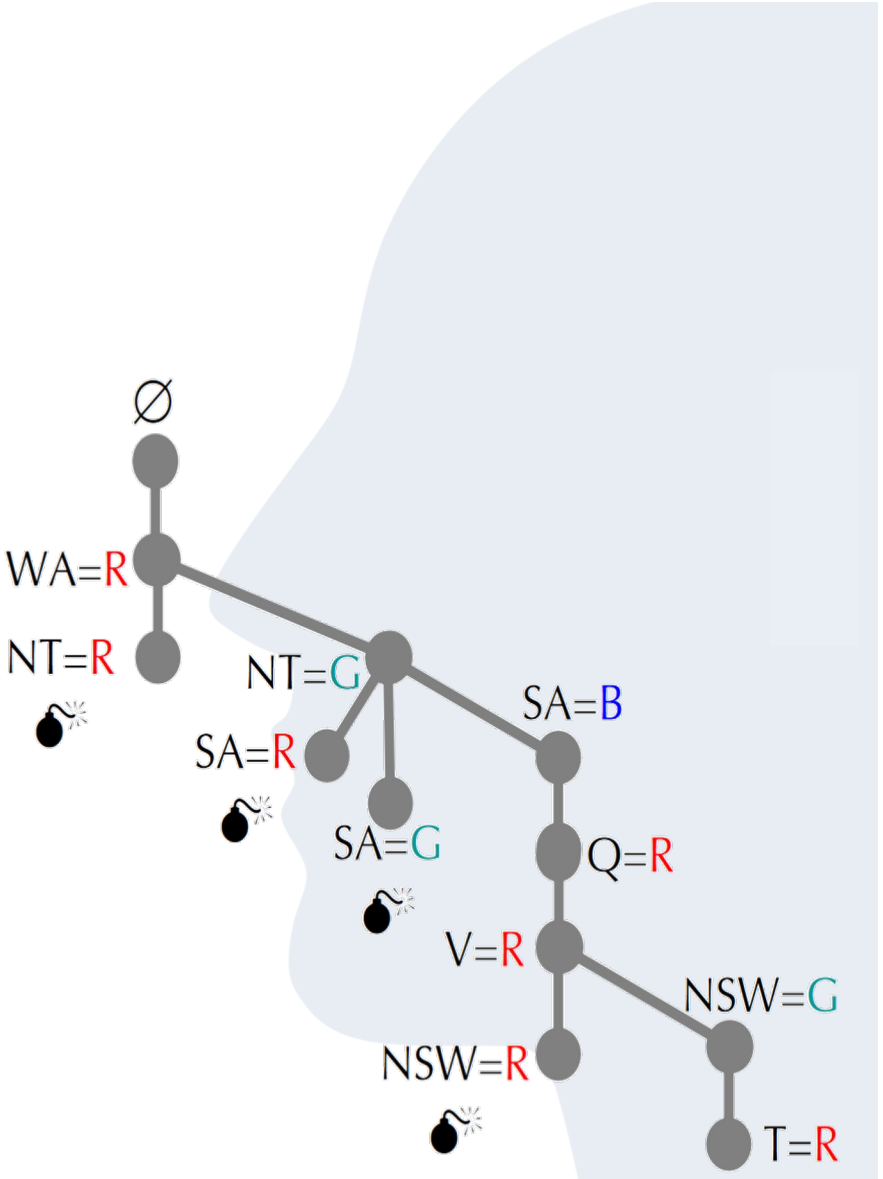


https://www.youtube.com/watch?v=Zq4upTEaQyM

Backtracking Search

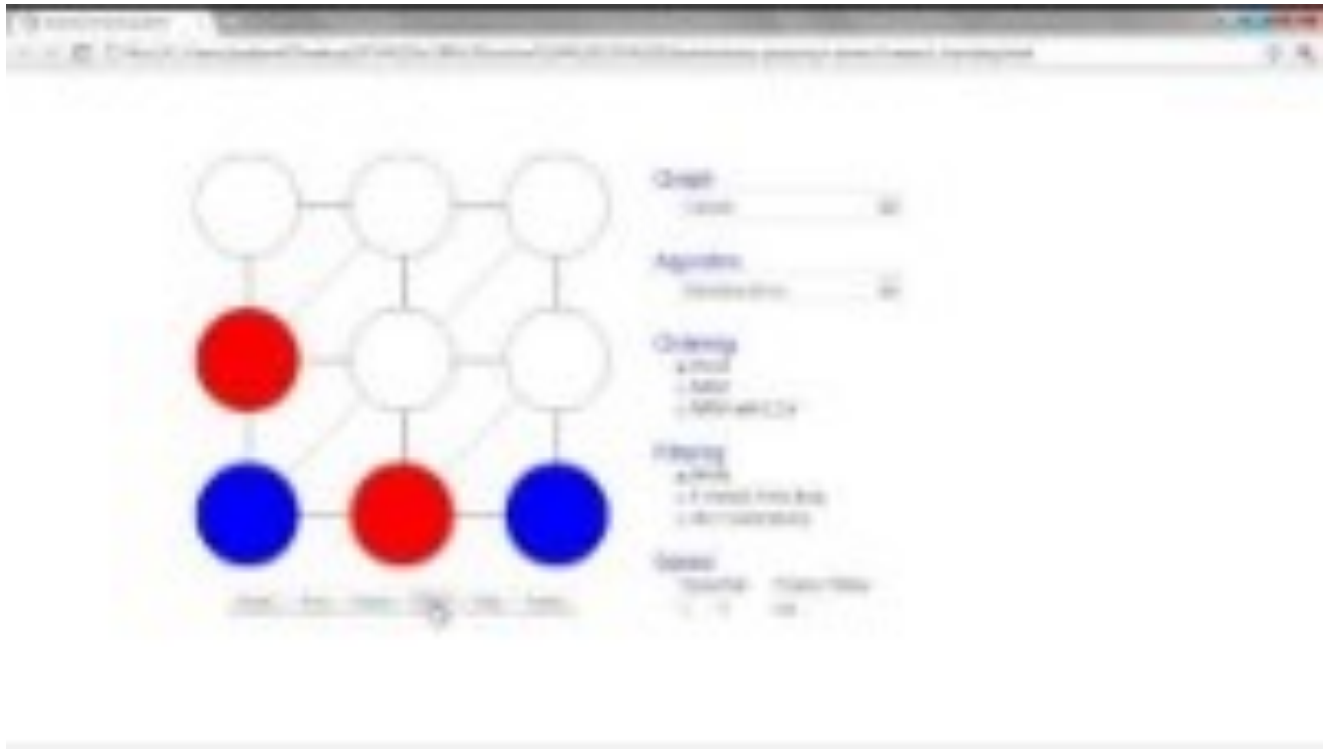
General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failures early?
4. Can we take advantage of problem structures?
5. …



Graph taken from J. Hertzberg, Uni Osnabrück

Backtracking Search
Example



https://youtu.be/Hv_JlWld9iQ?t=2560

Heuristics for CSPs
How do we select Heuristics

Domain-Specific Heuristics

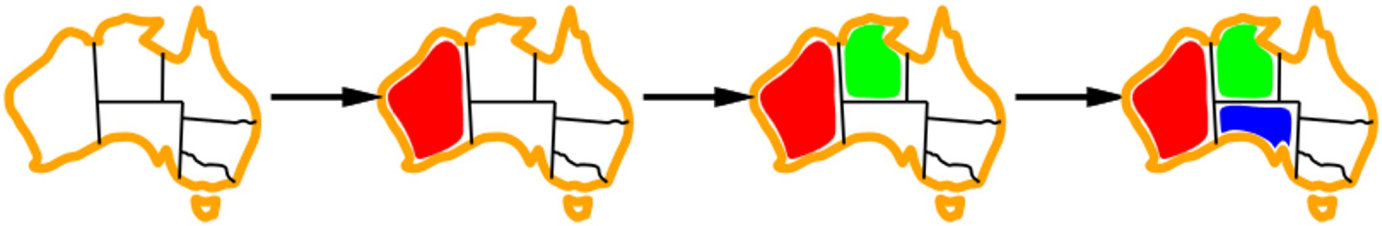
- Depend on the particular characteristics of the problem

General-purpose heuristics

- For CSP, good general-purpuse heuristics are known:

  Mininum Remaining Values Heuristic

  - choose the variable with the fewest consistent values



Heuristics for CSPs
How do we select Heuristics

Domain-Specific Heuristics

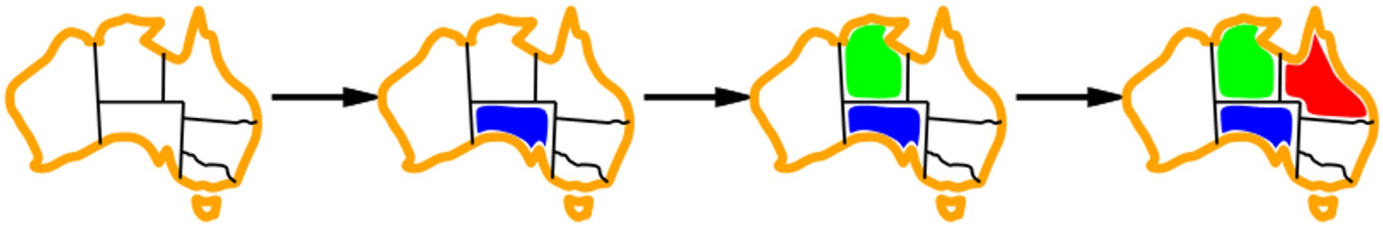
- Depend on the particular characteristics of the problem

General-purpose heuristics

- For CSP, good general-purpuse heuristics are known:

  Degree Heuristic

  - choose the variable with the most constraints on remaining variables



Heuristics for CSPs
How do we select Heuristics

Domain-Specific Heuristics

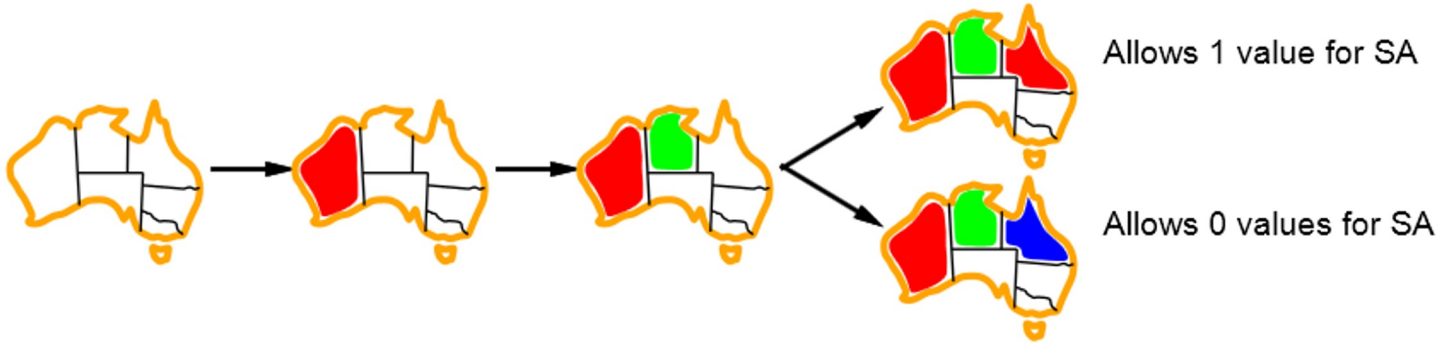
- Depend on the particular characteristics of the problem

General-purpose heuristics

- For CSP, good general-purpuse heuristics are known:

  Least Constraining Value Heuristic

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



Heuristics for CSPs
How do we select Heuristics

Domain-Specific Heuristics

- Depend on the particular characteristics of the problem

General-purpose heuristics

- For CSP, good general-purpuse heuristics are known:

  Mininum Remaining Values Heuristic
  - choose the variable with the fewest consistent values

  Degree Heuristic
  - choose the variable with the most constraints on remaining variables

  Least Constraining Value Heuristic
  - Given a variable, choose the value that rules out the fewest values in the remaining variables

Used in this order, these three can greatly improve search speed

Constraint Propagation
On the Example of Sudoku

Problem: CSP with 81 variables

Constraints:

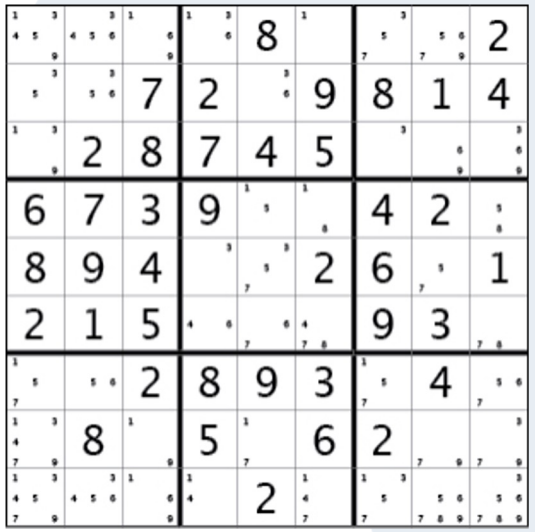
- some values are assigned in the start
- 27 constraints on 9 values that must all be different (9 rows, 9 columns, 9 squares)

Constraint Propagation

- People often write a list of possible values into empty fields
- try to successively eliminate values

Status

- Automated constraint solvers can solve the hardest puzzles in no time



Constraint Propagation
Node Consistency

Node Consistency

A variable is consistent if the possible values of this variable are conform to all unary constraints.

Local Consistency

Local consistency is defined by a graph where each node in it is consistent with its neighbors. This is done by iteratively enforcing the constraints corresponding to the edges.

Example for Node Consistency

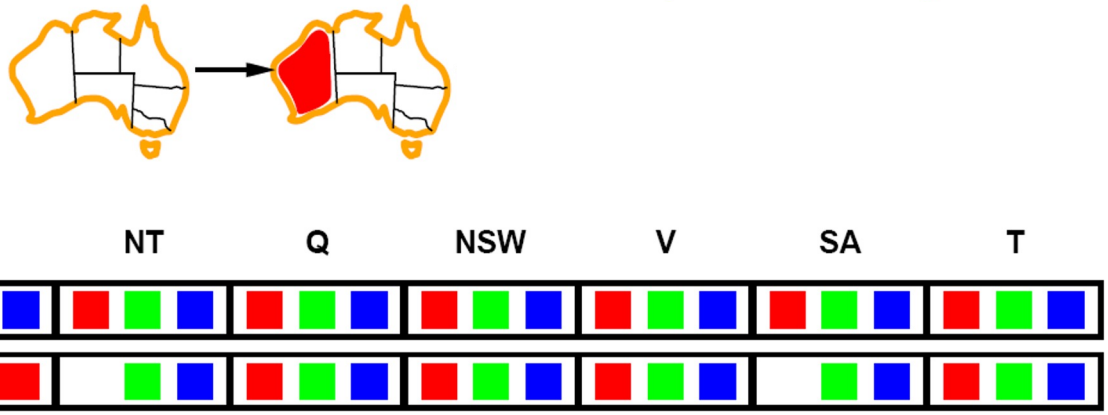
Sudoku → Some nodes are already filled out, i.e., constrained to a single value

Forward Checking

Idea: Keep track of remaining legal values for unassigned variables and terminate search when any variable has no more legal values

Forward Checking

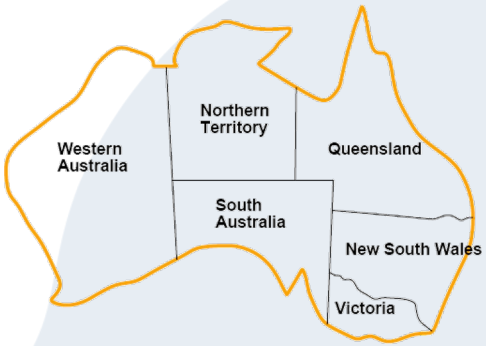
Idea: Keep track of remaining legal values for unassigned variables and terminate search when any variable has no more legal values



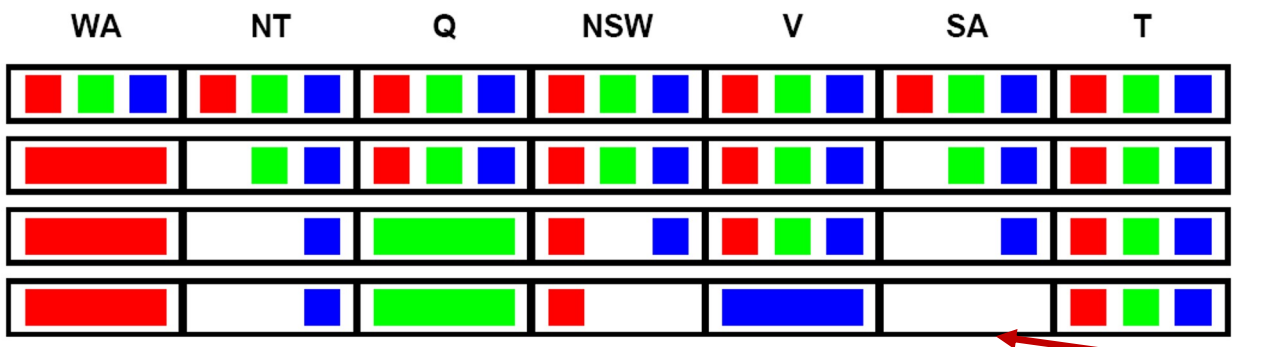
Forward Checking

Idea: Keep track of remaining legal values for unassigned variables and terminate search when any variable has no more legal values

Forward Checking

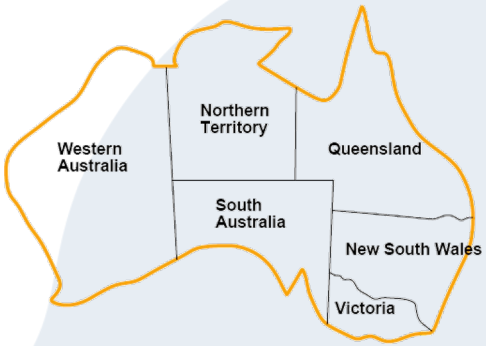


Idea: Keep track of remaining legal values for unassigned variables and terminate search when any variable has no more legal values



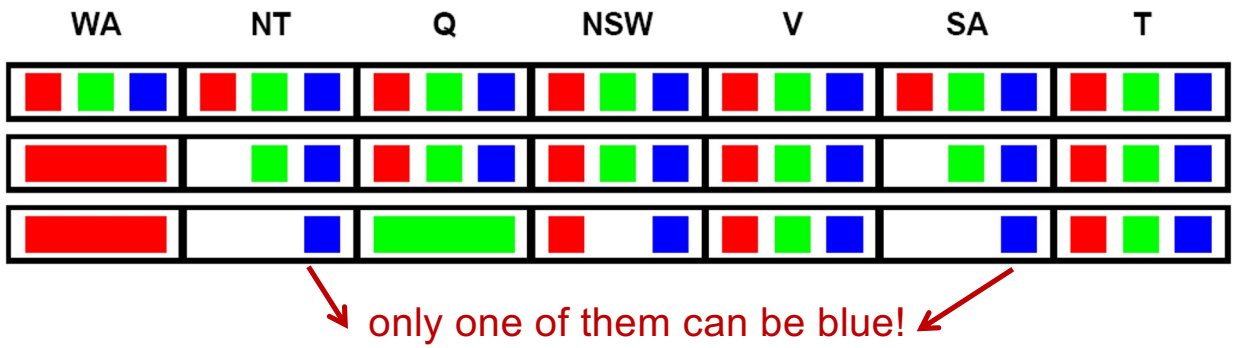
no further assignment possible!

Forward Checking



Idea: Keep track of remaining legal values for unassigned variables and terminate search when any variable has no more legal values

One step earlier, we could have seen already the conflict!



Arc Consistency

Let us improve performance by looking at larger sets of constraints and how they interact

Arc

A constraint involving two variables is called arc or binary constraint (see Slide 12

Arc Consistency

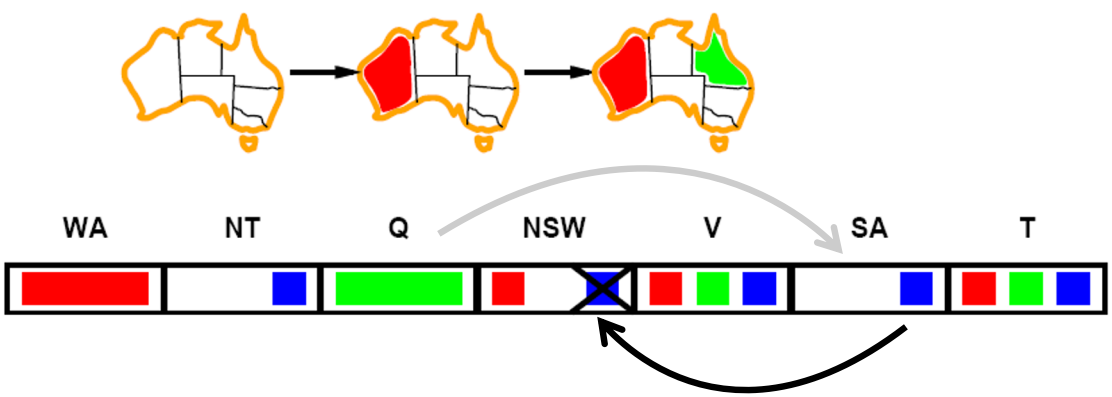
An arc is called consistent if for each value of X in the domain of X there is a value Y in the domain of Y such that the constraint *arc(X,Y)* is satisfied.

$$\forall X \in dom(X), \exists Y \in dom(Y) \text{ such that } arc(X, Y) \text{ is satisfied}$$

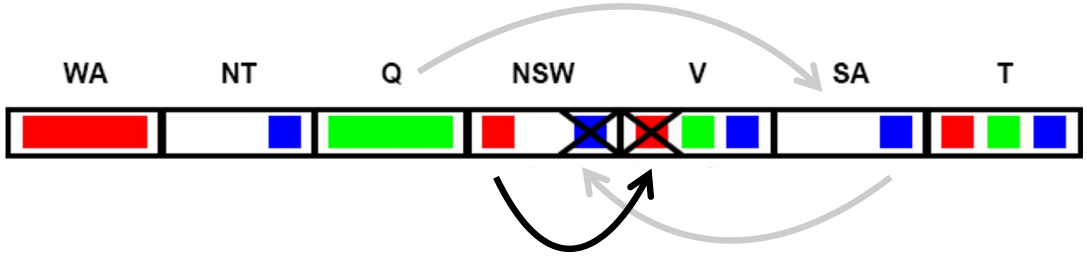
- Can be generalized to n-ary constraints
  - each tuple involving the variable X_i has to be consistent

Maintaining Arc Consistency (MAC)

After each new assignment of a value to a variable, possible values of the neighbors have to be updated:



If one variable (NSW) looses a value (blue), we need to recheck its neighbors as well because they might have lost a possible value



Arc Consistency
AC-3 Algorithm

```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., 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) ← 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
```

If X loses a value, neigbors of X need to be rechecked.

```
function REMOVE-INCONSISTENT-VALUES( X_i, X_j) returns true iff succeeds
  removed ← 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 ↔ X_j
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

Path Consistency

Arc Consistency is often sufficient to…

- solve the problem (all variable domains are reduced to size 1)
- show that the problem cannot be solved (some domains empty)

…but may not be enough

- there is always a consistent value in the neighboring region

Path Consistency

- tightens the binary constraints by considering triples of values

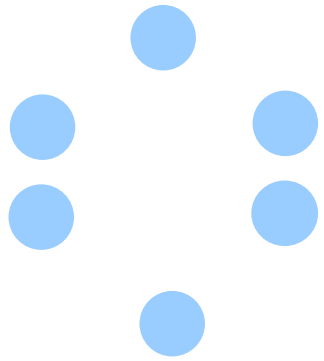
A pair of variables (X_i , X_j) is path-consistent with X_m if

- for every assignment that satisfies the constraint on the arc (X_i ,X_j)
- there is an assignment that satisfies the constraints on the arcs (X_i,X_m) and (X_j,X_m)

k-Consistency

The concept can be generalized so that a set of k values need to be consistent

- 1-consistency = node consistency
- 2-consistency = arc consistency
- 3-consistency = path consistency
- …



- May lead to faster solution but checking for *k*-Consistency is exponentional in *k* in the worst case
- Therefore arc consistency is most frequently used in practice

Path Consistency
Example Sudoku

Simple puzzles can be solved with AC-3

- The puzzle has 9 constraints on the rows, 9 on the columns and 9 on the square (27 in total)
  - each such constraint requires that 9 values are all different
- These 9-valued AllDiff constraints can be converted into pairwise binary constraints
  - 9x8/2 = 36 pairwise constraints
- Therefore 27x36 = 972 arc constraints

However, not all problems can be solved with constraint progapagation alone

To solve all types of puzzles we need a bit of search

Constraint Propagation and Backtracking Search

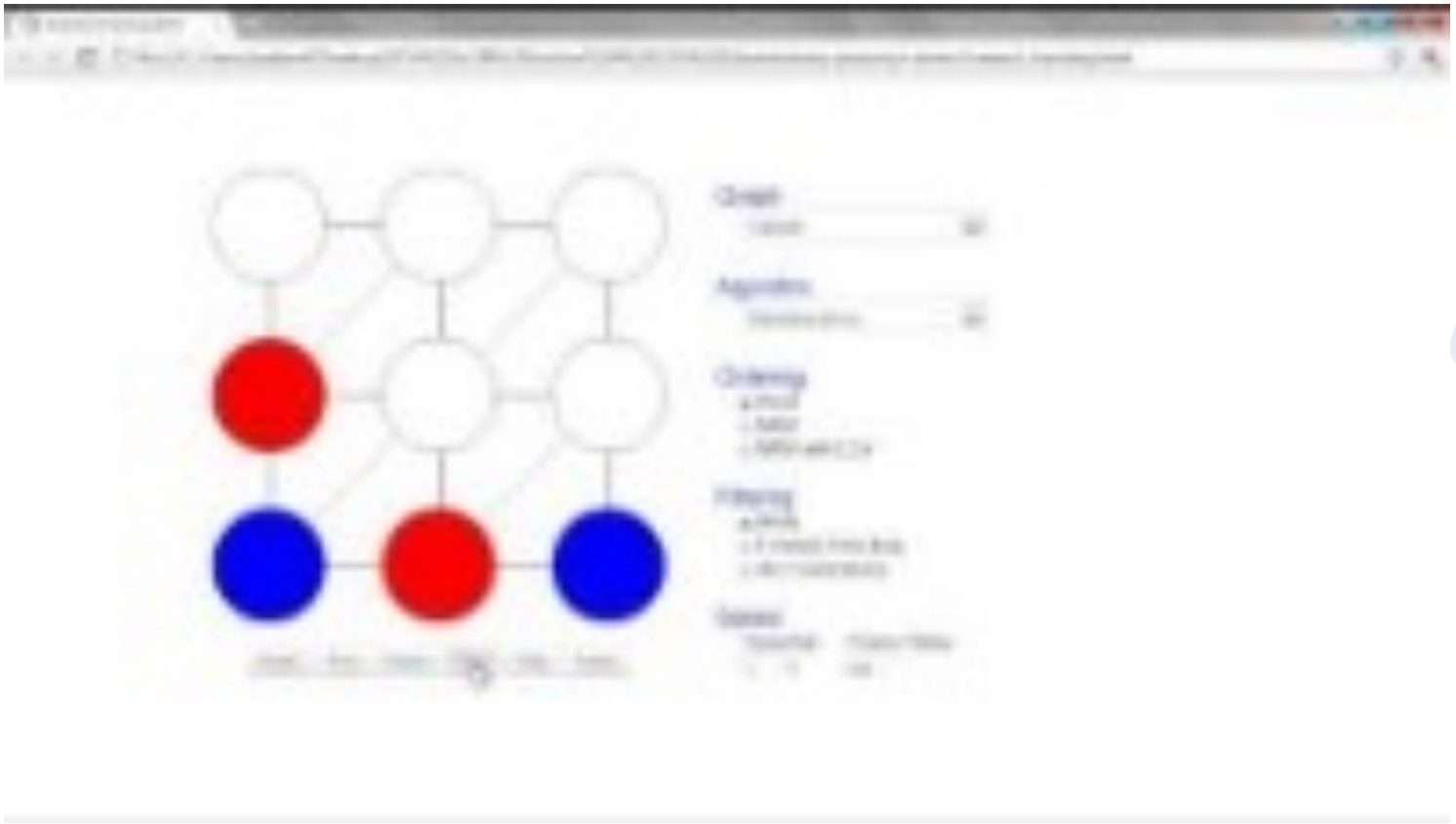
Idea: Each time a variable is assigned, a constraint propagation algorithm is run in order to reduce the number of choice points in the search

- Can improve the speed of backtracking search further

Possible Algorithm

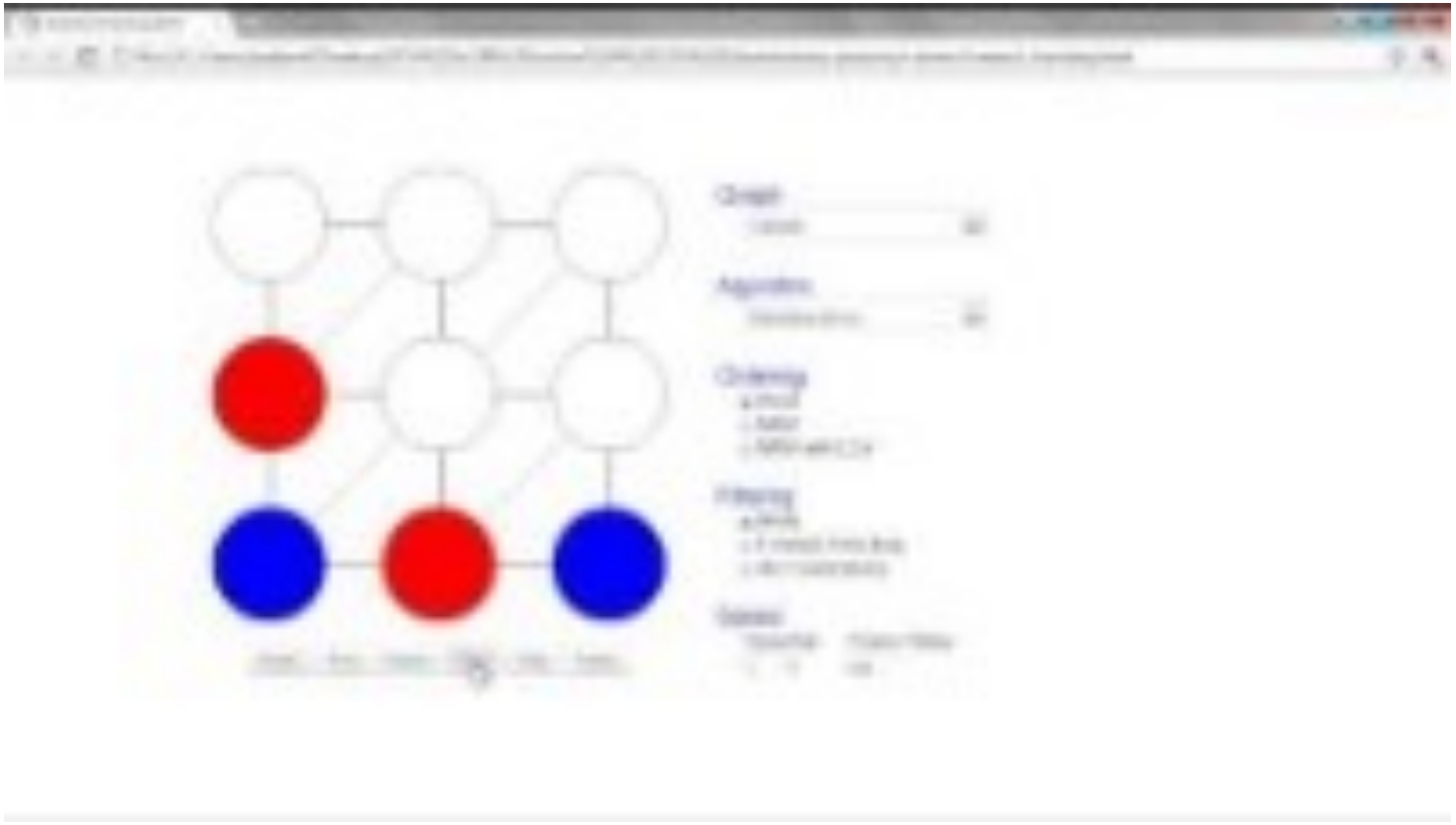
- Forward Checking
- AC-3, initial queue of constraints only contains constraints with the variable that has been changed

Illustration of Constraint Propagation + Search
Example 1



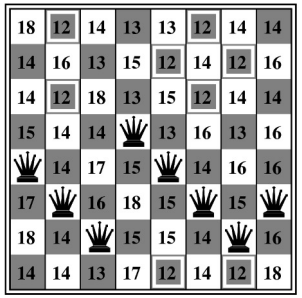
https://youtu.be/Hv_JlWld9iQ?t=2945

Illustration of Constraint Propagation + Search
Example 2 (more complex tree)



https://youtu.be/Hv_JlWld9iQ?t=3887

Local Search for CSPs



Modifications for CSPs:

- work with complete states
- allow states with unsatisfied constraints
- operators reassign variable values

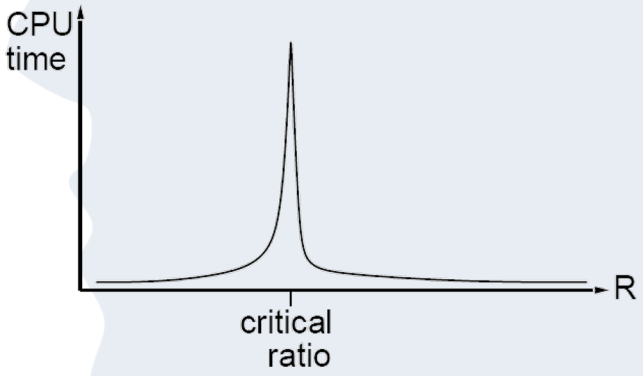
Min-conflicts Heuristic:

- randomly select a conflicted variable
- choose the value that violates the fewest constraints
- hill-climbing with h(n) = # of violated constraints

Performance:

- can solve randomly generated CSPs with a high probability
- except in a narrow range *R*

Min-conflicts is the heuristic that we studied for the 8-queens problems.



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

The Power of Problem Decomposition

Assumption: Search space for a constraint satisfaction with n variables, each of which can have d values = $O(d^n)$

Idea: Decomposing the problem into subproblems with c variables each

- Each problem has complexity = $O(d^c)$
- There are n/c such problems → Total complexity = $O(n/c \cdot d^c)$
- But unconditional independence is rare!

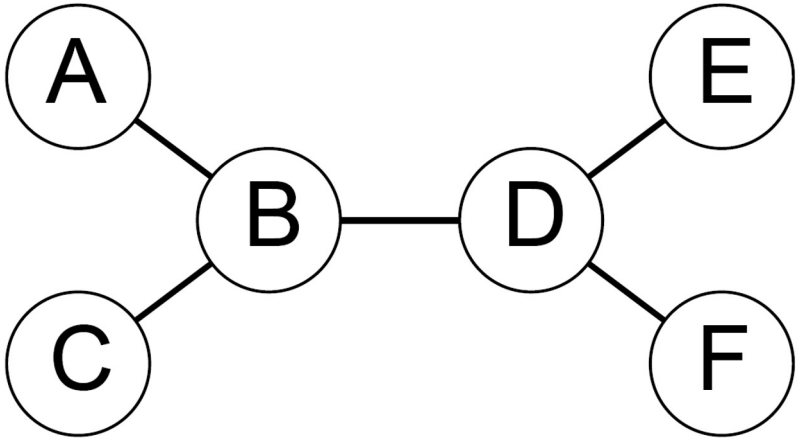
Effect: The total complexity can be reduced from exponential in n to linear in n (assuming that c is a constant parameter)!

$$2^{80} \neq 4 \cdot 2^{20}$$

Example: n = 80, d = 2, c = 20 →

Tree-structured CSPs

A CSP is tree-structured if in the constraint graph any two variables are connected by a single path

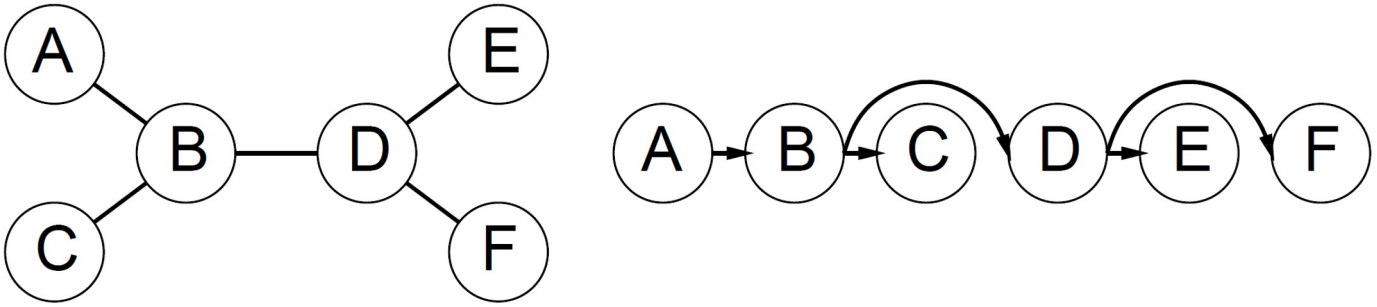


Theorem: Any tree-structured CSP can be solved in linear time in the number of variables (more precisely: $O(n \cdot d^2)$)

Tree-structured CSPs

Linear Algorithm for Tree-Structured CSPs

1. Choose a variable as a root, order nodes so that a parent always comes before its children (each child can have only one parent)
2. For $j = n$ downto 2
   - Make the arc (X_i , X_j) arc-consistent, calling REMOVE-INCONSISTENT-VALUE(X_i , X_j)
3. For $i = 1$ to n
   - Assign to X_i any value that is consistent with its parent.



Tree-structured CSPs
Nearly Tree-structured Problems

Problem: Tree-structures problems are also rare…

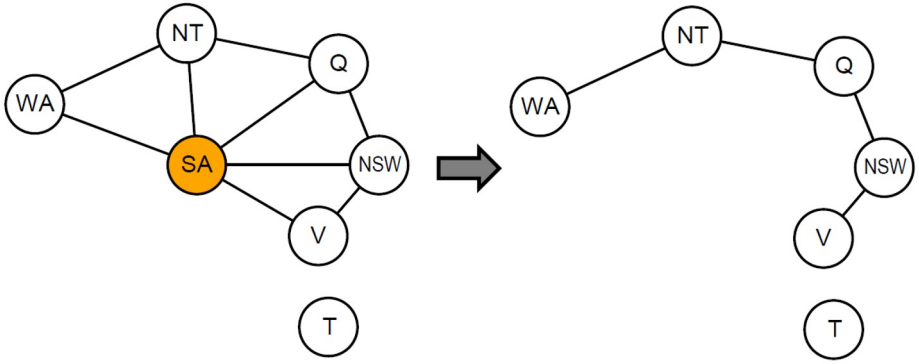
Two approaches for making problems tree-structured:

1. **Cutset Conditioning**
   - Removing nodes so that the remaining nodes form a tree
2. **Collapsing nodes together**
   - Decompose the graph into a set of independent tree-shaped subproblems

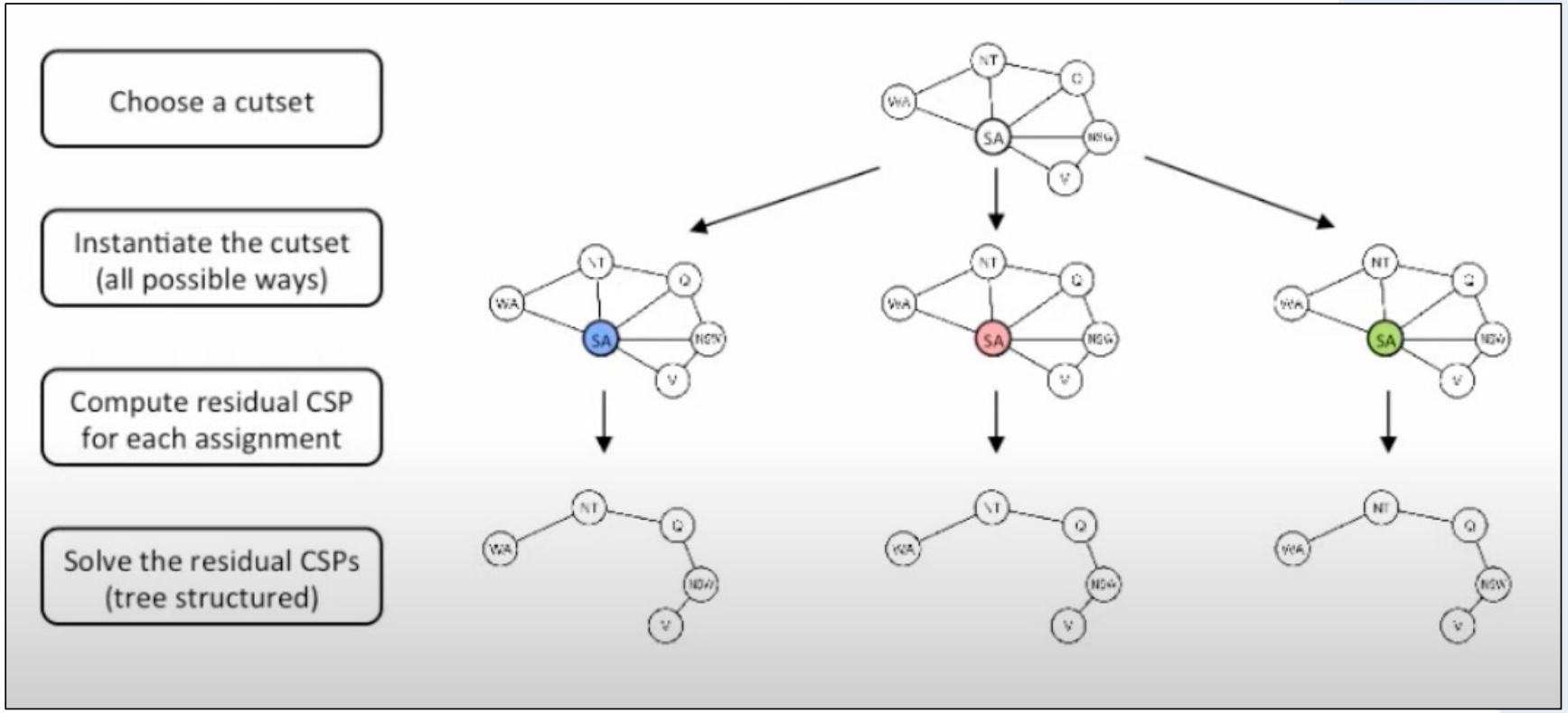
Tree-structured CSPs
Cutset Conditioning

1. Choose a subset *S* of the variables such that the constraint graph becomes a tree after removal of *S* (= the cycle cutset)
2. Choose a (consistent) assignment of variables for *S*
3. Remove from the remaining variables all values that are inconsistent with the variables of *S*
4. Solve the CSP problem for the remaining variables
5. If no solution → choose a different assignment for variables in 2)

Example: *S*={SA}



Tree-structured CSPs
Cutset Conditioning



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
- The CSP representation allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time

Summary

Definition of a Constraint Satisfaction Problems (CSP)

CSPs and Search

Backtracking Search

Constraint Propagation

Forward Checking

Arc Consistency

Tree-structured CSPs

You should be able to:

- Describe a CSP and its features
- Build a backtracking search
- Give information how to improve backtracking search
- Describe the goal of forward checking and Constraint Propagation
- Describe possible ways to reduce the complexity of CSPs

Next Week: Logic 1