Constraint Satisfaction

Ch. 6.1–6.4 (skip 6.3.3))

Based in part on slides by: Marie desJardin, Paula Matuszek, Luke Zettlemoyer, Dan Klein, Pieter Abbeel, Dan Weld, Stuart Russell, Andrew Moore

1

Bookkeeping

- HW2 out tonight
- Last time: local search, intro to constraint satisfaction problems
- Today: CSPs!

2

From Last Time…

- Standard search problems:
  - State is a "black box": arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything
- Constraint satisfaction problems (CSPs) are special subset of search problems
- General Idea
  - View a problem as a set of variables
  - To which we have to assign values
  - That satisfy a number of (problem-specific) constraints

3

Today's Class

- What's a Constraint Satisfaction Problem (CSP)?
  - A.K.A., Constraint Processing / CSP paradigm
- How do we solve them?
  - Algorithms for CSPs
- Search Terminology

Constraint (n): A relation … between the values of one or more mathematical variables (e.g., x>3 is a constraint on x).

Constraint satisfaction assigns values to variables so that all constraints are true.

– *http://foldoc.org/constraint*

4

Real-World Problems

- Scheduling
- Temporal re
- Building des
- Planning
- Optimizatio
- Vision
- Graph layout
- management
- nguage processing
- r biology /
- gn

Assignment problems
Timetabling problems
Hardware configuration
Gate assignment in airports
Space Shuttle Repair
Transportation scheduling
Factory scheduling
…

5

Where we are so far

- Standard search problems:
  - State is a "black box": arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything
- Constraint satisfaction problems (CSPs): A special subset of search problems
  - State is defined by **variables**
  - With values from a **domain** D
  - Goal test is a set of **constraints** specifying allowable combinations of values for subsets of variables
- CSPs formulation allows for special **optimized algorithms**
- A good example of trading generality for utility (in this case, speed)

6

Constraint Satisfaction

- Con•straint /kənˈstrānt/, (noun):
  - Something that limits or restricts someone or something.[1]
  - A relation … between the values of one or more mathematical variables (e.g., x>3 is a constraint on x), that assigns values to variables so that all constraints are true.[2]
- In search, constraints exist on?
- General Idea
  - View a problem as a set of variables
  - To which we have to assign values
  - That satisfy a number of (problem-specific) constraints

[1] Merriam-Webster online.
[2] The Free Online Computing Dictionary.

7

Overview

- **Constraint satisfaction**: a problem-solving paradigm
- Constraint programming, constraint satisfaction problems (**CSPs**), constraint logic programming…
- Algorithms for CSPs
  - Backtracking (systematic search)
  - Constraint propagation (k-consistency)
  - Variable and value ordering heuristics
  - Backjumping and dependency-directed backtracking

8

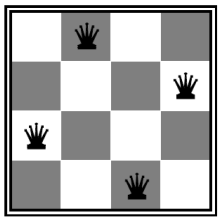
Search Vocabulary

- We've talked about caring about *goals* (end states) vs. *paths*
- These correspond to…
  - **Planning:** finding sequences of actions
    - Paths have various costs, depths
    - Heuristics to guide, frontier to keep backup possibilities
    - Examples: chess moves; 8-puzzle; homework 2
  - **Identification:** assignments to variables representing unknowns
    - The goal itself is important, not the path
    - Examples: Sudoku; map coloring; N queens; scheduling; planning
- CSPs are specialized for identification problems

9

Slightly Less Informal Definition of CSP

- **CSP** = Constraint Satisfaction Problem
- Given:
  - A finite set of **variables**
  - Each with a **domain** of possible values they can take (often finite)
  - A set of **constraints** that limit the values the variables can take on
- **Solution**: an assignment of values to variables that satisfies all constraints.



10

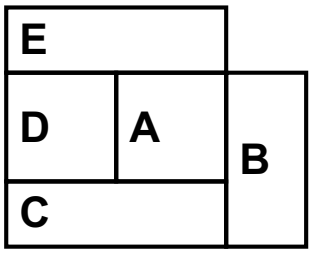
CSP Applications

- Decide if a solution exists
- Find some solution
- Find all solutions
- Find the "best solution"
  - According to some metric (objective function)
  - Does that mean "optimal"?

11

Informal Example: Map Coloring

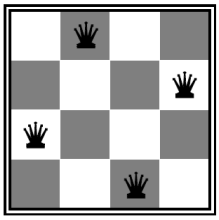
- Given a 2D map, it is always possible to color it using three colors
- Such that:
  - No two adjacent regions are the same color
  - Start thinking: What are the values, variables, constraints?



12

Slightly Less Informal

- Constraint satisfaction problems (CSPs): a special subset of search problems where…
- **State** is defined by variables X_i with values from a domain D
  - D may be finite
  - Sometimes D depends on i
- **Goal test** is a set of constraints specifying allowable combinations of values for variables

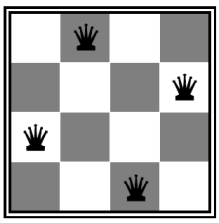


13

Example: N-Queens (1)

- Formulation 1:
  - Variables: X_{ij}
  - Domains: $\{0, 1\}$
  - Constraints:

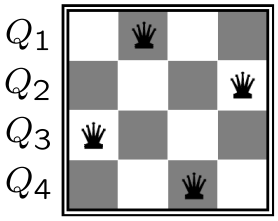
$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\sum_{i,j} X_{ij} = N$$



14

Example: N-Queens (2)

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
    - Actually, tuples of {(1–N, 1–N)}
  - Constraints:



Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

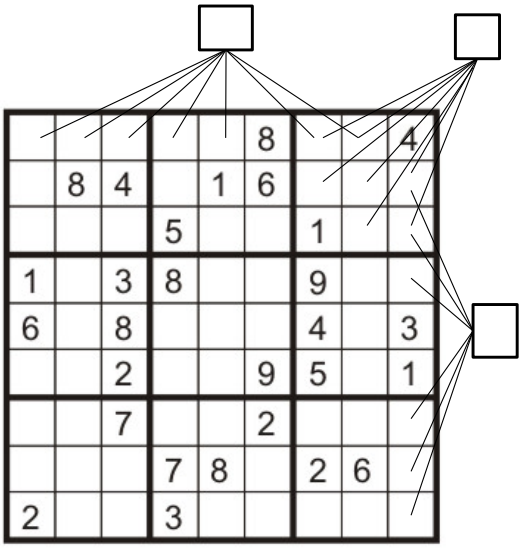
-or-

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

• • •

15

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:
  - 9-way alldiff for each column
  - 9-way alldiff for each row
  - 9-way alldiff for each region
  - (or can have a bunch of pairwise inequality constraints)

16

Example: SATisfiability

- Given a set of propositions containing variables, find an assignment of the variables to {false, true} that satisfies them.

Special case!

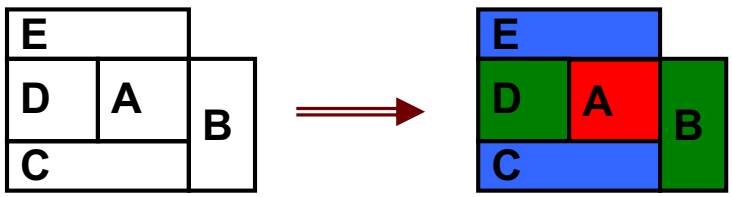
- For example, the clauses:
  - $(A \vee B \vee \neg C) \wedge (\neg A \vee D)$
  - (equivalent to $(C \rightarrow A) \vee (B \wedge D \rightarrow A)$)
- are satisfied by
  - A = false
  - B = true
  - C = false
  - D = false

Variables:	propositional variables
Domains:	{T, F}
Constraints:	logical formula

17

Exercise: Map Coloring II

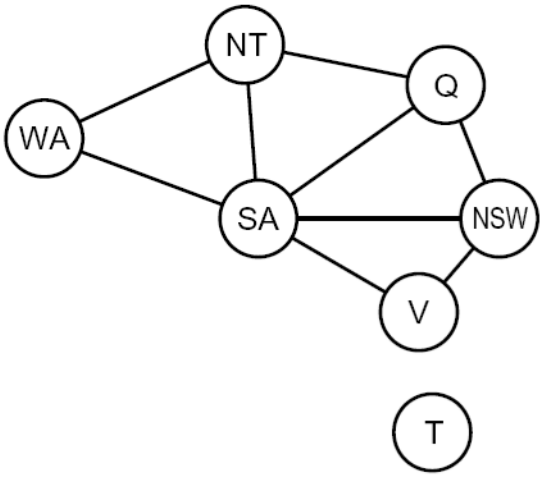
- Variables:
- Domains:
- Constraints:
- One solution:



18

Constraint Graph (Network)

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints



19

Formal Definition: Constraint Network (CN)

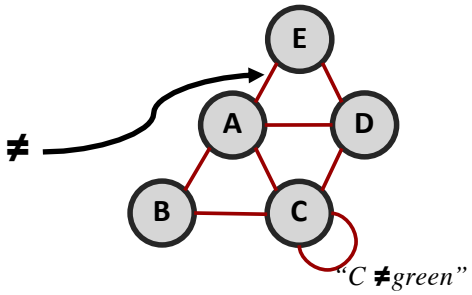
A **constraint network** (CN) consists of

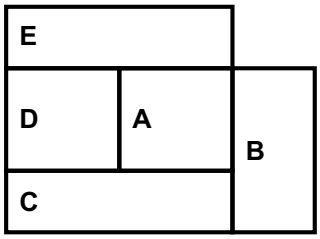
- A set of variables $X = \{x_1, x_2, \ldots x_n\}$
- Each with an associated domain of values $\{d_1, d_2, \ldots d_n\}$.
  - The domains are typically finite
- A set of constraints $\{c_1, c_2 \ldots c_m\}$ where
  - Each constraint defines a **predicate,** which is a **relation** over some subset of X.
  - E.g., c_i involves variables $\{X_{i1}, X_{i2}, \ldots X_{ik}\}$ and defines the relation $R_i \subseteq D_{i1} \times D_{i2} \times \ldots D_{ik}$

20

Constraint Restrictions

- **Unary** constraint: only involves one variable
  - e.g.: C can't be green.
- **Binary** constraint: only involves two variables
  - e.g.: E ≠ A





21

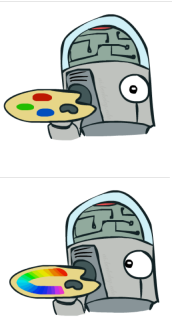
Formal Definition of a CN (cont.)

- An **instantiation** is an assignment of a value $d_x \in D$ to some subset of variables S.
  - Any assignment of values to variables
  - Ex: $Q_2 = \{2,3\} \wedge Q_3 = \{1,1\}$ **instantiates** Q_2 and Q_3
- An instantiation is **legal** iff it does not violate any constraints
- A **solution** is an instantiation of all variables
  - A **correct solution** is a **legal** instantiation of all variables

22

Variations: Variables

- Discrete Variables
  - Finite domains
    - Size d means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by linear programming



23

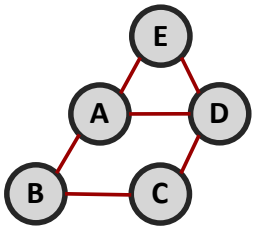
Variations: Constraints

- Unary constraints involve a single variable (equivalent to reducing domains)
- Binary constraints involve pairs of variables
- Higher-order constraints involve 3 or more variables: e.g., cryptarithmetic column constraints
- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a **cost** for each variable assignment
  - Gives constrained optimization problems
  - (We'll ignore these until we get to Bayes' nets)

24

Typical Tasks for CSP

- Solutions:
  - Does a solution exist?
  - Find one solution
  - Find all solutions
- Transform the CN into an equivalent CN that is easier to solve
- Given a partial instantiation, can we do these?



25

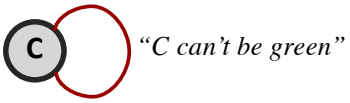
Binary CSP

- **Binary CSP**: all constraints are binary or unary
- Can convert a non-binary CSP → binary CSP by:
  - Introducing additional variables
  - One variable per constraint
  - One binary constraint for each pair of original constraints that share variables
- "Dual graph construction"

26

Binary CSPs: Why?

- Can always represent a binary CSP as a **constraint graph** with:
  - A **node** for each variable
  - An **arc** between two nodes iff there is a constraint on the two variables
  - Unary constraint appears as a self-referential arc



27

Exercise: Sudoku

- Variables
- Domains
- Blocks

v_{11}	3	v_{13}	1
v_{21}	1	v_{23}	4
3	4	1	2
v_{41}	v_{42}	4	v_{44}

28

Running Example: Sudoku

- Constraints (implicit or intensional)
  - $C^R : \forall i, \cup_j v_{ij} = D$
    (every value appears in every row)
  - $C^C : \forall j, \cup_i v_{ij} = D$
    (every value appears in every column)
  - $C^B : \forall k, \cup (v_{ij} \mid ij \in B_k) = D$
    (every value appears in every block)

v_{11}	3	v_{13}	1
v_{21}	1	v_{23}	4
3	4	1	2
v_{41}	v_{42}	4	v_{44}

29

Running Example: Sudoku

All binary constraints!

- Possible representation: pairwise inequality
  - $I^R : \forall i, j \neq j' : v_{ij} \neq v_{ij'}$
    (no value appears twice in any row)
  - $I^C : \forall j, i \neq i' : v_{ij} \neq v_{i'j}$
    (no value appears twice in any column)
  - $I^B : \forall k, ij \in B_k, i'j' \in B_k, ij \neq i'j' : v_{ij} \neq v_{i'j'}$
    (no value appears twice in any block)

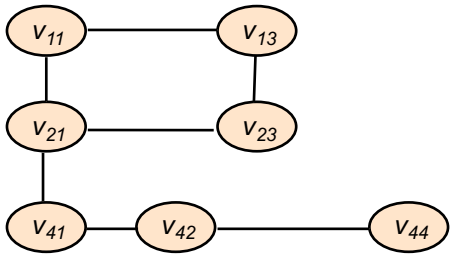
v_{11}	3	v_{13}	1
v_{21}	1	v_{23}	4
3	4	1	2
v_{41}	v_{42}	4	v_{44}

30

Exercise: Draw the Sudoku CN

1. $I^R : \forall i, j \neq j' : v_{ij} \neq v_{ij'}$ (no value appears twice in any row)
2. $I^C : \forall j, i \neq i' : v_{ij} \neq v_{i'j}$ (no value appears twice in any column)
3. $I^B : \forall k, ij \in B_k, i'j' \in B_k, ij \neq i'j' : v_{ij} \neq v_{i'j'}$ (no value appears twice in a block)

v_{11}	3	v_{13}	1
v_{21}	1	v_{23}	4
3	4	1	2
v_{41}	v_{42}	4	v_{44}



31

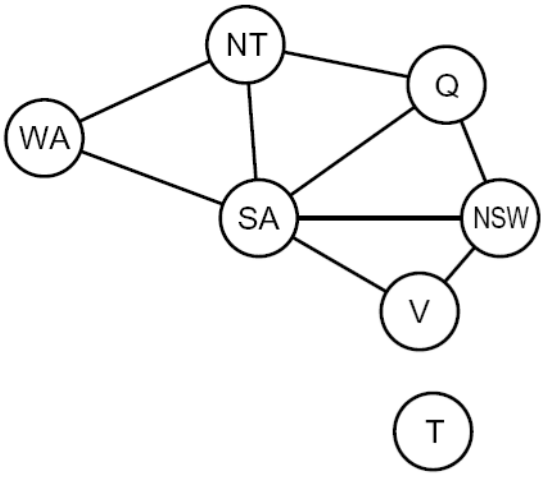
Solving Constraint Problems

- Systematic search
  - Generate and test
  - Backtracking
- Constraint propagation (consistency)
- Variable ordering heuristics
- Value ordering heuristics
- Variable elimination
- Backjumping and dependency-directed backtracking

32

CSPs as Search

We already know how to deal with searching graphs!



States?

Operators?

Initial State?

Goal Test?

33

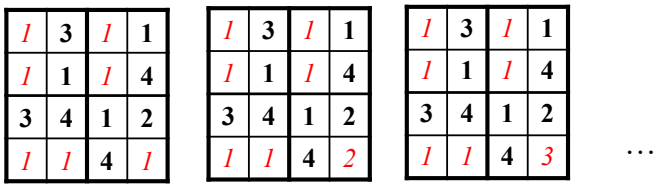
Standard Search Formulation

- Standard search formulation of CSPs (incremental)
- Let's start with a straightforward, dumb approach, then fix it
- **States** are defined by the **values assigned so far**
  (ex: WA=red, T=red is a state)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

34

Super Naïve: Generate and Test

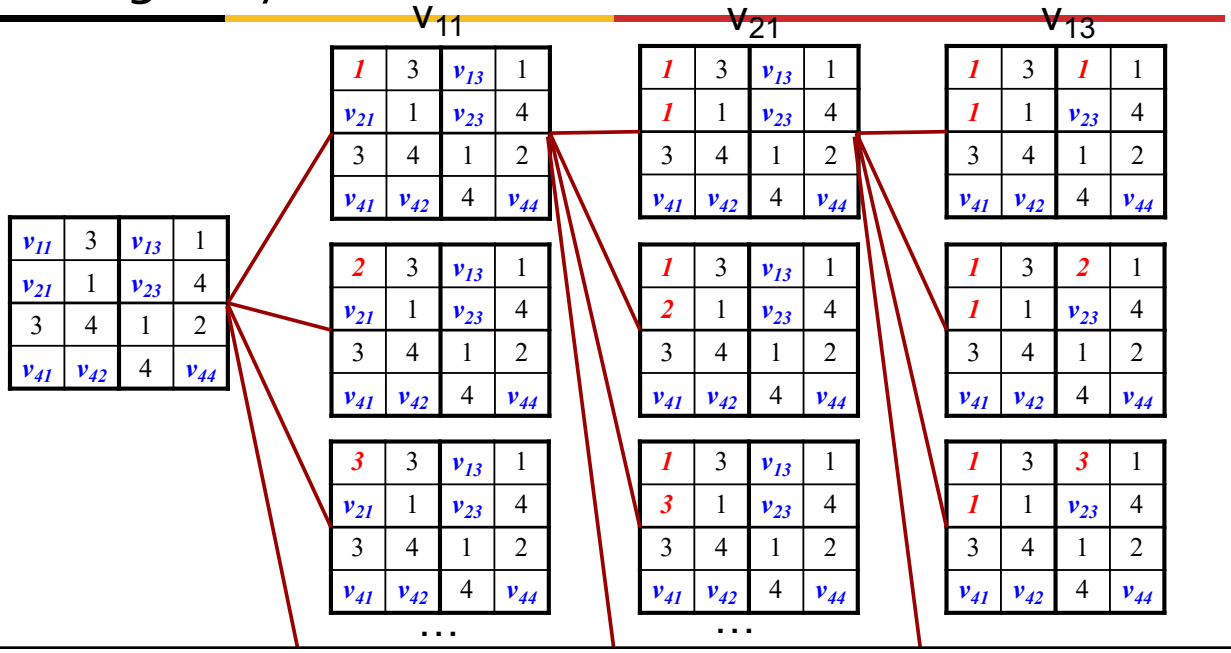
- Try every possible assignment of domain elements to variables until you find one that works:



- Doesn't check constraints until all variables have been instantiated
- Very inefficient way to explore the space of possibilities (4^7 for this trivial Sudoku puzzle, mostly illegal)

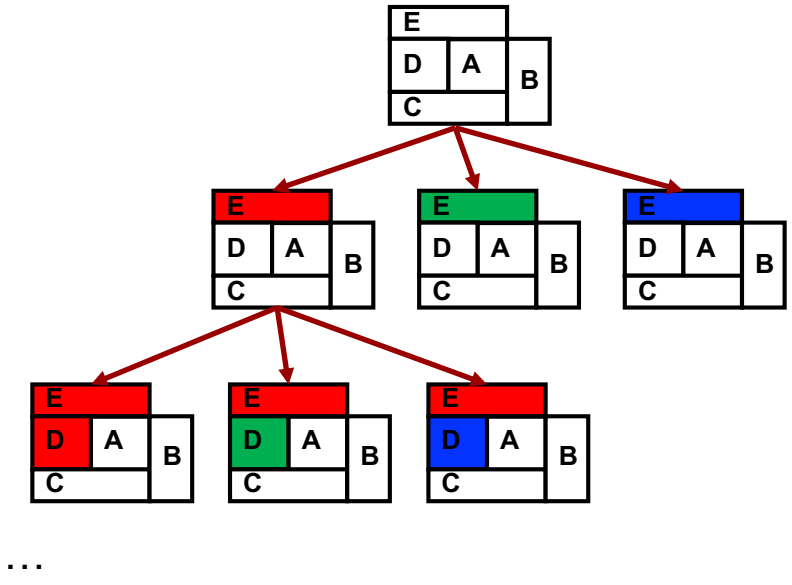
35

Marginally Better: DFS



36

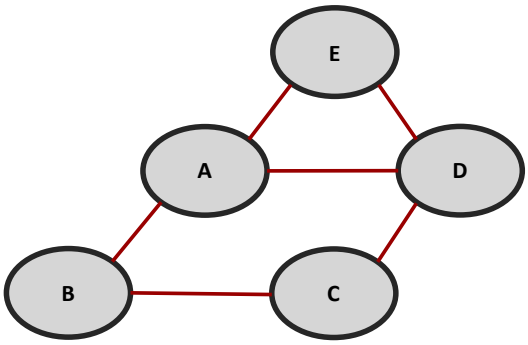
Search: DFS



37

Sidenote: Consistency

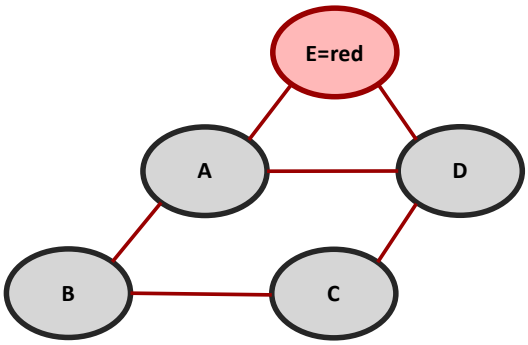
- The goal is to find a solution that is consistent with (doesn't violate) constraints
- An instantiation (assignment of values to variables) is said to be **consistent** if no constraints are violated



38

Consistency

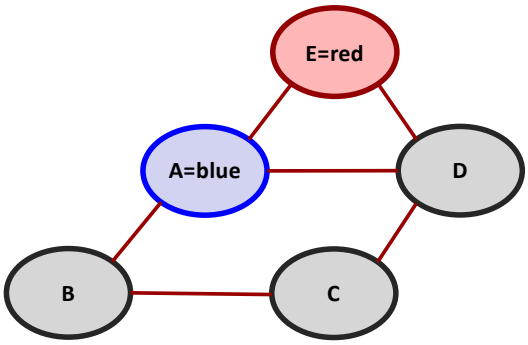
- The goal is to find a solution that is consistent with (doesn't violate) constraints
- An instantiation (assignment of values to variables) is said to be **consistent** if no constraints are violated



39

Consistency

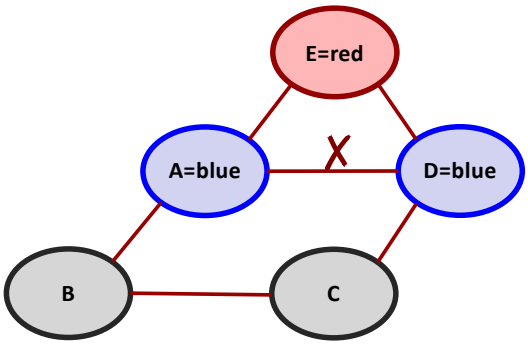
- The goal is to find a solution that is consistent with (doesn't violate) constraints
- An instantiation (assignment of values to variables) is said to be **consistent** if no constraints are violated



40

Consistency

- The goal is to find a solution that is consistent with (doesn't violate) constraints
- An instantiation (assignment of values to variables) is said to be **consistent** if no constraints are violated



41

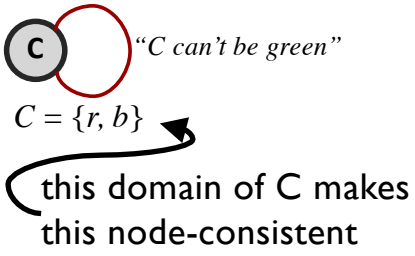
Consistency

- Once the whole graph is consistent, we have a solution (a legal instantiation of values to all variables)
- There are multiple **kinds** of consistency
- Different kinds give us different guarantees for performance and correctness

42

Node Consistency

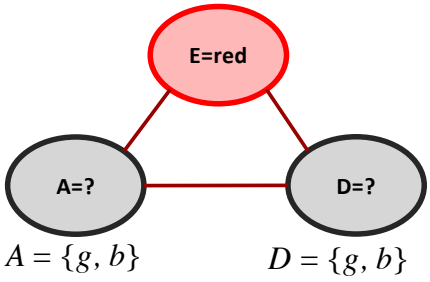
- **Node consistency:** every value in **node X**'s domain (every value we think it might take) is consistent with *X's unary constraints*
  - A graph is node-consistent if all nodes are node-consistent
  - Let's say C can't be green
  - C = {red, ~~green~~, blue}



43

Arc Consistency

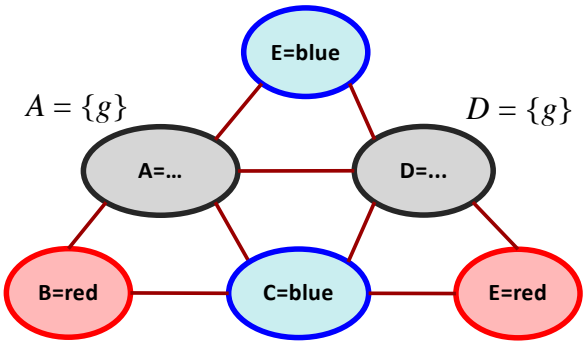
- **Arc consistency:** A variable in a CSP is **arc-consistent** if every value in its domain satisfies the variable's binary constraints
- For every value *x* of X in Arc(X,Y):
  - **∃*y*** for Y
  - That satisfies the constraint represented by the arc
- A graph is arc-consistent if all arcs are arc-consistent



44

Arc Consistency: Example

- For every value x of X in Arc(X,Y):∃y for Y that satisfies the constraint represented by the arc

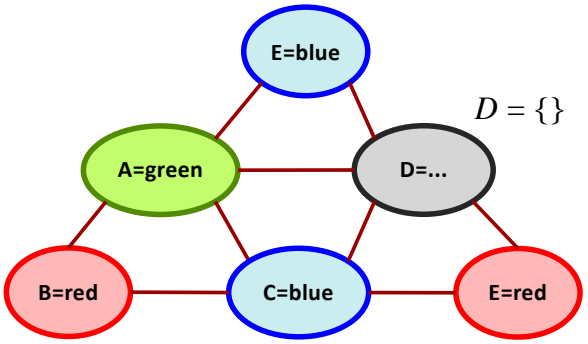


- Is this instantiation arc-consistent?
- So far, yes!

45

Arc Consistency: Example

- For every value x of X in Arc(X,Y):∃y for Y that satisfies the constraint represented by the arc



- Is this instantiation arc-consistent?
- Not any more!

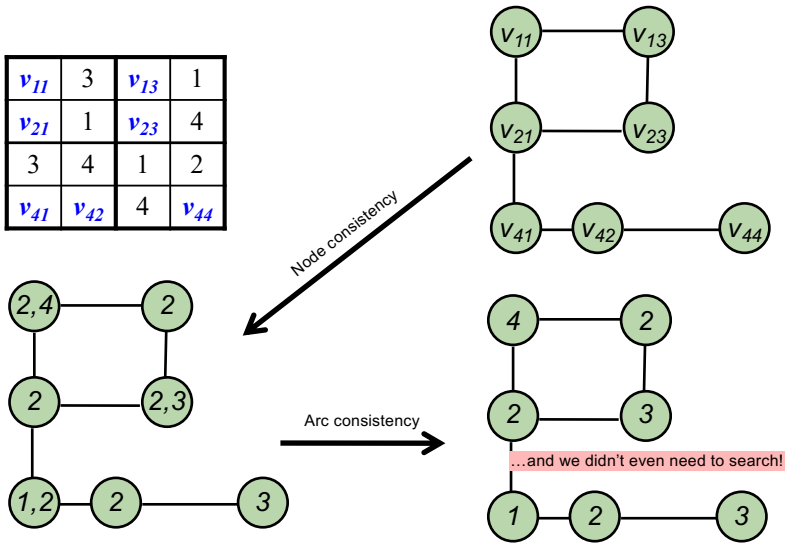
46

Next Up: Constraint Propagation

- To create arc consistency, we perform **constraint propagation**: that is, we repeatedly reduce the domain of each variable to be consistent with its arcs
- How do we find a set of consistent assignments?
- Constraints reduce # of **legal values for a variable**
  - Which may then reduce legal values of another variable
- Key idea: **local consistency**
  - Enforce nearby constraints
  - Propagate

47

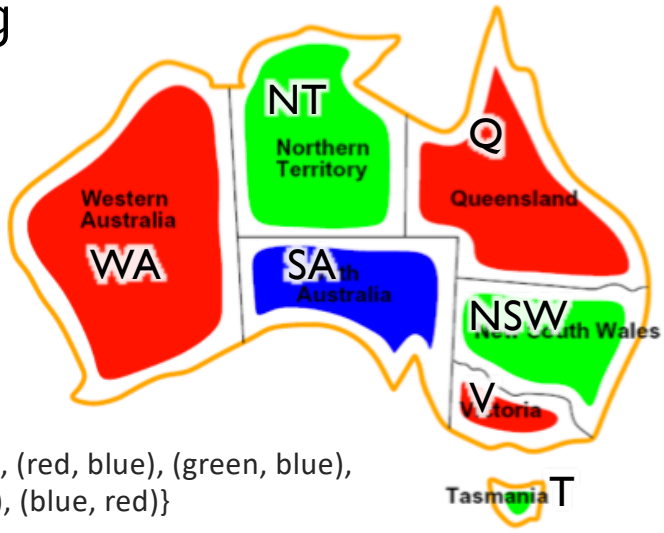
Constraint Propagation: Sudoku



48

Example: Map Coloring

- Variables:
  - WA, NT, Q, SA, NSW, V, T
- Domain:
  - D = {red, green, blue}
- Constraints: adjacent regions must have different colors
  - Implicit: WA ≠ NT
  - Explicit: (WA, NT) ∈ {(red, green), (red, blue), (green, blue), (green, red), (blue, red)}

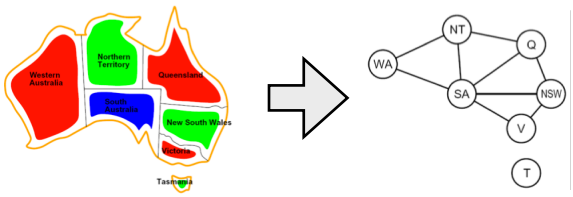


- Solutions are assignments satisfying all constraints, e.g.:
  - {WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green }

49

Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints

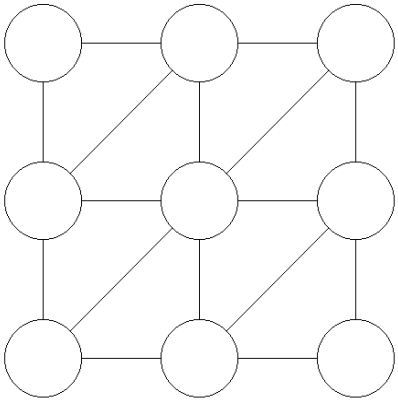


- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!

50

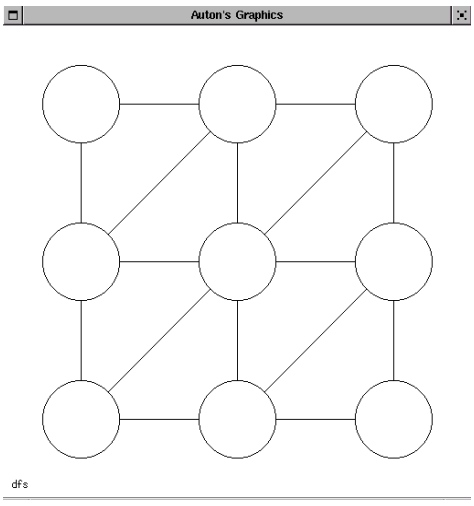
Search Methods

- Map Coloring
- How does DFS do?
- How does BFS do?



51

DFS: not good!



52

Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- **Idea 1: Only consider a single variable at each point**
  - Variable assignments are commutative, so fix the ordering
    - [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- **Idea 2: Only allow fully legal assignments at each point**
  - Consider only values which do not conflict with existing assignments
  - Might have to do some computation to figure out whether a value is ok
  - "Incremental goal test"
- Depth-first search with these two improvements is called **backtracking search**
  - We *backtrack* when there's no legal assignment for the next variable

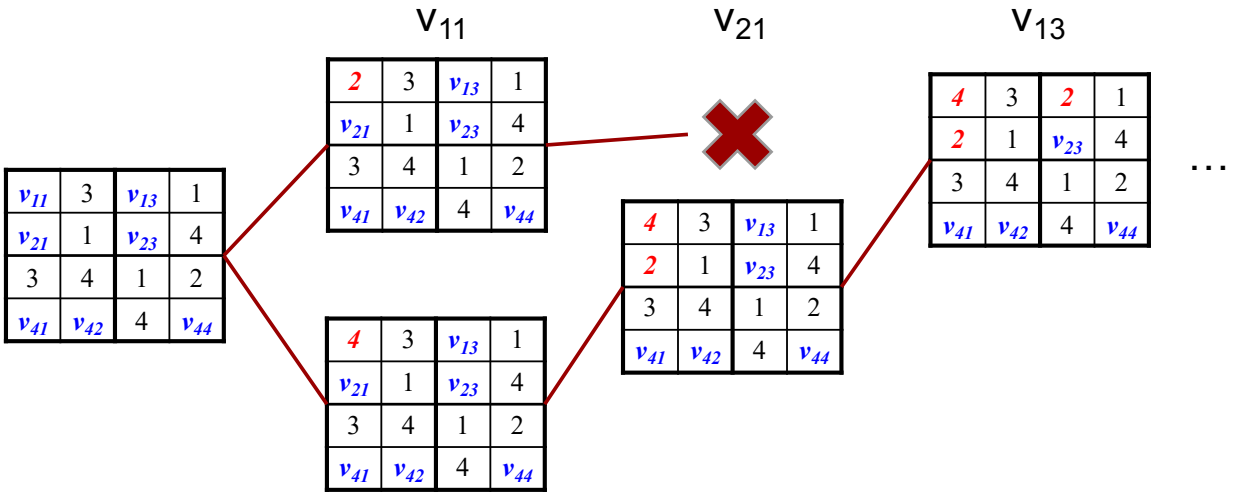
53

Systematic Search: Backtracking (a.k.a. DFS)

- Consider the variables in some order
  - Pick an unassigned variable
  - Give it a provisional value
  - **That is consistent with all of the constraints** (← this is new!)
- If no such assignment can be made, we've reached a dead end and need to backtrack to the previous variable
- Continue this process until:
  - A solution is found, or
  - We backtrack to the initial variable and have exhausted all possible values

54

Backtracking: Sudoku

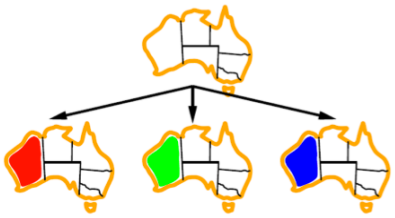


55

Backtracking Example

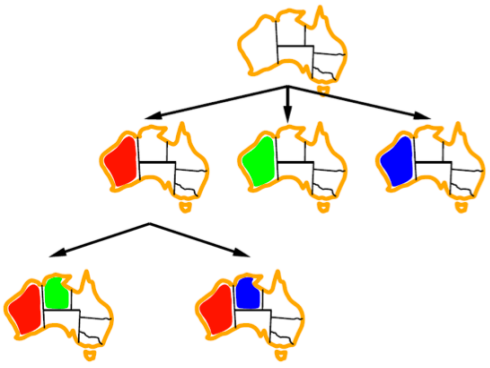
56

Backtracking Example



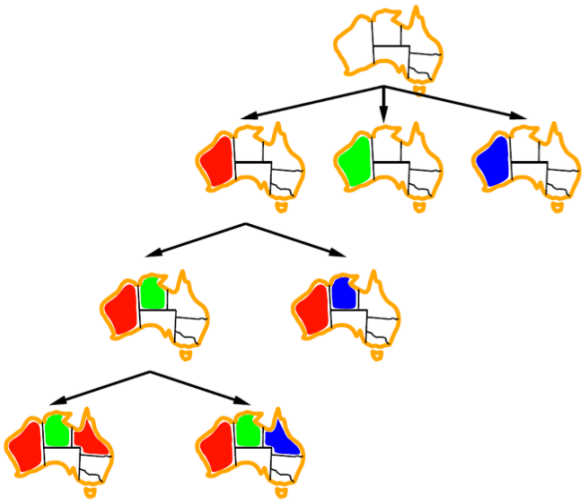
57

Backtracking Example



58

Backtracking Example



59

Backtracking Search

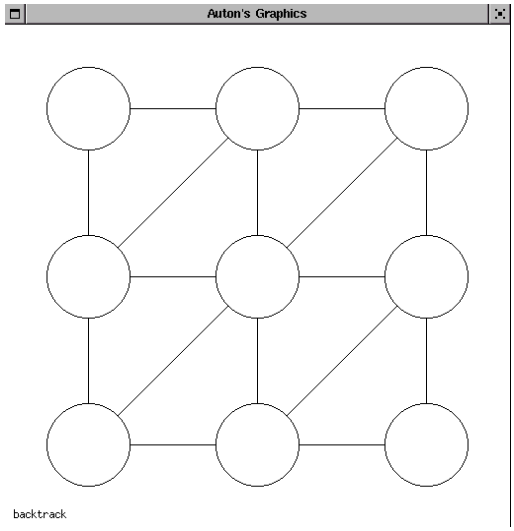
- What are the choice points?

```
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
```

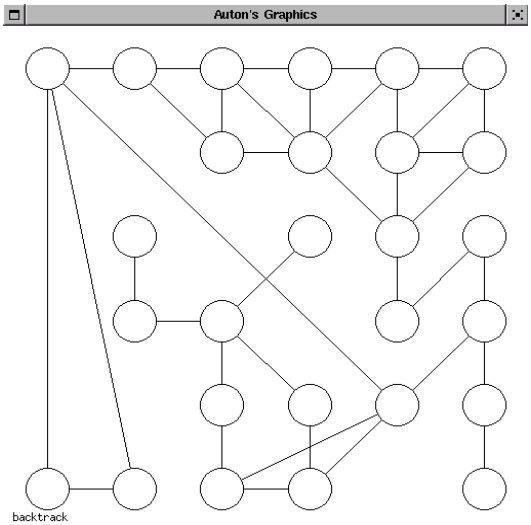
60

Backtracking



61

Good Enough?



62

Problems with Backtracking

- Thrashing: keep repeating same failed variable assignments
  - Consistency checking can help
  - Intelligent backtracking schemes can also help
- Inefficiency: can spend time exploring areas of search space that aren't likely to succeed
  - **Variable ordering can help**
  - IF there's a meaningful way to order them

v_{11}	3	v_{13}	1
v_{21}	1	v_{23}	4
3	4	1	2
v_{41}	v_{42}	4	v_{44}

63

Improving Backtracking

General-purpose ideas give huge gains in speed!

- **Ordering**: (queueing function ++)
  - Which variable should be assigned next?
  - In what order should its values be tried?
- **Filtering**: Can we detect inevitable failure early?
- **Structure**: Can we exploit the problem structure?

64

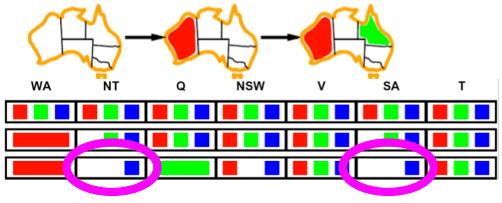
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment

65

Forward Checking

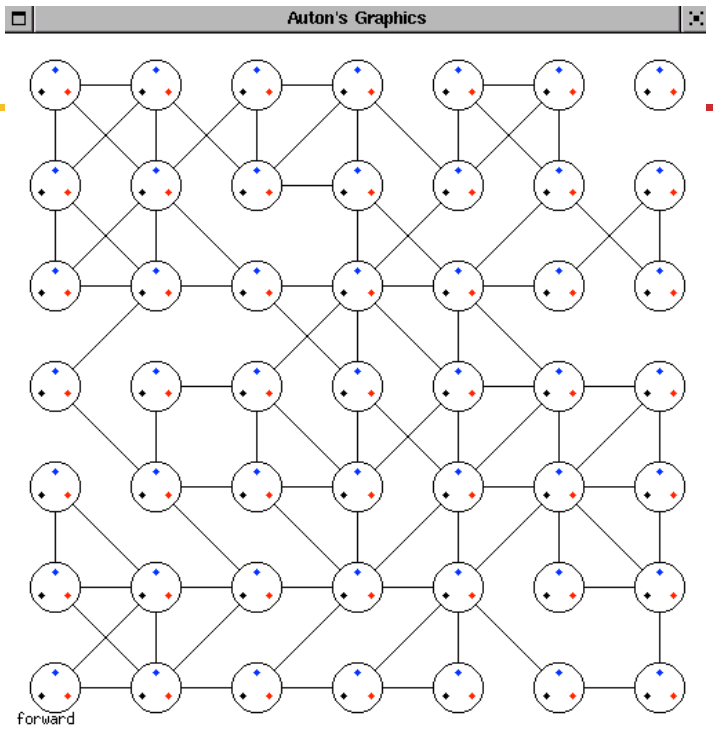
- Propagates information from assigned to **adjacent** unassigned variables
- Doesn't detect more distant failures



- NT and SA cannot both be blue! Why didn't we detect this?
  - It doesn't catch when ***two unassigned variables*** have no consistent assignment
- Constraint propagation enforces constraints **locally**
  - This is a local maximum!

66

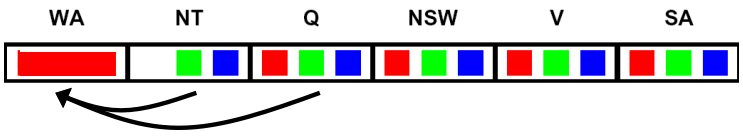
Are We Done?



67

Consistency of a Single Arc

- Simplest form of propagation makes each arc consistent
  - An arc X → Y is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constraint

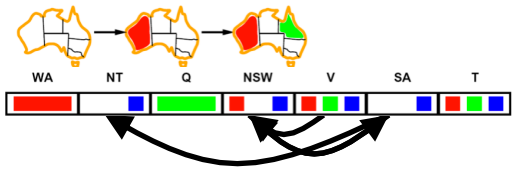


- Forward checking is enforcing consistency of **arcs pointing to each new assignment**

68

Better: Arc Consistency of an **Entire CSP**

- A simple form of propagation makes sure all arcs are consistent:

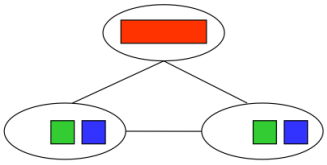


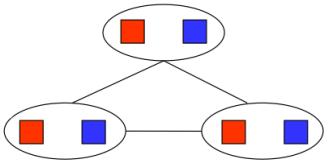
- If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure **earlier** than forward checking
- Can be run as a preprocessor or after each assignment

69

Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)
- Even with Arc Consistency you still need backtracking search!
  - Could run at every step of that search
  - Usually better to run it **once, before search**





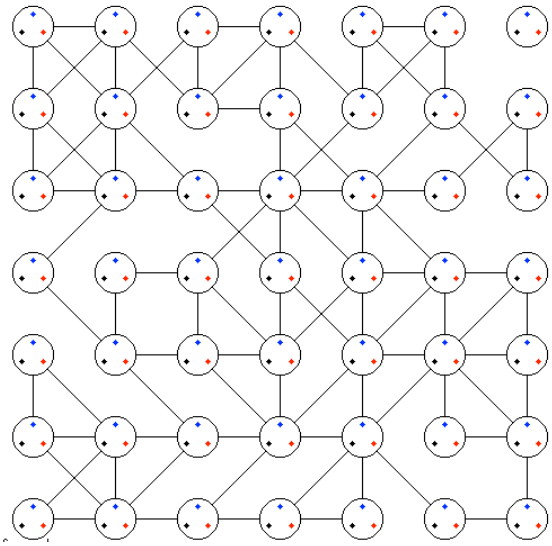
What went wrong here?

70

Approaches so far

- Generate-and-test
- DFS
- Backtracking
- Forward checking
- Arc consistency

- What else could we try?



71

K-consistency

- K-consistency generalizes the notion of arc consistency to **sets of more than two variables**
  - **Node Consistency** (1-Consistency): Each **single** variable's domain has a value which meets that variables unary constraints
  - **Arc Consistency** (2-Consistency): For each **pair** of variables, any consistent assignment to one can be extended to the other
  - **Path Consistency** (3-Consistency): For every **set of 3** vars, any consistent assignment to 2 of the variables can be extended to the third var
  - **K-Consistency**: For each k nodes, any consistent assignment to k-1 can be extended to the k^{th} node
- Higher k more expensive to compute!

72

Why Do We Care?

- Strong k-consistency: also k-1, k-2, ... 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
  - Choose any assignment to any variable
  - Choose a new variable
  - By 2-consistency, there is a choice consistent with the first
  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - ...
- A strongly N-consistent CSP with N variables can be solved **without backtracking**
  - **If** we find an appropriate variable ordering!

Which leads us to **ordering**

73

Ordered Constraint Graphs

- Select a variable ordering, V_1, …, V_n
- **Width of a node** in this OCG is the number of arcs leading to *earlier* variables:
  - width(V_i) = count ((V_i, V_k) | k < i)
- **Width of the OCG** is the maximum width of any node:
  - width(OCG) = max (width (V_i)), 1 ≤ i ≤ N
- **Width of an unordered CG** is the minimum width of all orderings of that graph (best you can do)

74

Backtracking Search

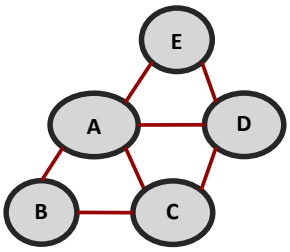
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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
```

75

Possible Variable Orderings

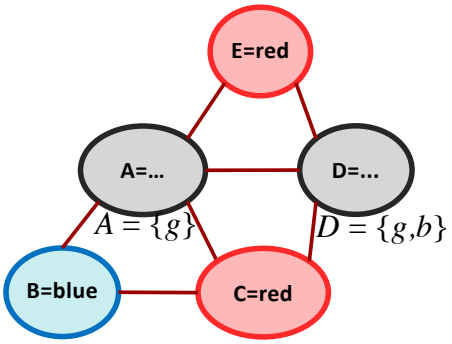
- **Intuition**: choose variables that are highly constrained early in the search process; leave easy ones for later.
- How?
  - **Minimum width ordering** (MWO): identify OCG with minimum width
  - **Maximum cardinality ordering**: approximation of MWO that's cheaper to compute: order variables by decreasing cardinality (a.k.a. degree heuristic)



76

Ordering: Minimum Remaining Values

- **Fail first principle** (FFP): choose variable with fewest remaining values
- Implementation: **minimum remaining values** (MRV))
  - **Static** FFP: use domain size of variables
  - **Dynamic** FFP (search rearrangement method): At each choice, select the variable with the fewest *remaining* values



77

Ordering: Minimum Width

- Minimum remaining values (MRV):
  - Choose the variable with the *fewest remaining* legal values

  - Why min rather than max?
- Also called "most constrained variable"
- "Fail-fast" ordering

78

Ordering: Maximum Degree

- Tie-breaker among MRV variables
  - What is the very first state to color? (All have 3 values remaining.)
- Maximum degree heuristic
  - Choose the variable participating in the most constraints on remaining variables

- Why most rather than fewest constraints?

79

Variable Orderings II

- **Maximal stable set**: find largest set of variables with no constraints between them, save these for last
- **Cycle-cutset tree creation**: Find a set of variables that, once instantiated, leave a tree of uninstantiated variables; solve these, then solve the tree without backtracking
- **Tree decomposition**: Construct a tree-structured set of connected subproblems

80

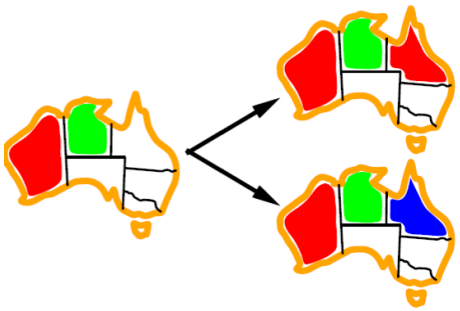
Value Ordering

- **Intuition**: Choose **values** that are the **least** constrained early on, leaving the most legal values available for later variables
  - **Maximal options method** (a.k.a. **least-constraining-value** heuristic): Choose the value that leaves the most legal values for not-yet-instantiated variables
  - **Min-conflicts**: For iterative repair search (Coming up)
  - Symmetry: Introduce **symmetry-breaking constraints** to constrain search space to 'useful' solutions (don't examine more than one symmetric/isomorphic solution)

81

Ordering: Least Constraining Value

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the least constraining value
  - I.e., the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)



- Why least rather than most?

- **Combining these ordering ideas makes 1000 queens feasible**

82

Min-Conflicts Heuristic

- Iterative repair method
  1. Find some "reasonably good" initial solution
     - E.g., in N-queens problem, use greedy search through rows, putting each queen where it conflicts with the smallest number of previously placed queens, breaking ties *randomly*
  2. Pick a variable in conflict (randomly)
  3. Select a new value that *minimizes* the number of constraint violations
     - O(N) time and space
  4. Repeat steps 2 and 3 until done

Performance depends on quality and informativeness of initial assignment; inversely related to distance to solution

83

The Intuition for MRV, MD, LCV

- We want to enter the most promising branch, **but** we also want to detect failure quickly
- Minimum Remaining Values + Maximum Degree
  - Choose the variable that is **most likely to cause failure**
  - It must be assigned at some point, so if it is doomed to fail, better to find out soon
- Least Constraining Value
  - We hope our early value choices do not doom us to failure
  - Choose the value that is most likely to succeed

84

Iterative Repair

- Start with an initial complete (but probably invalid) assignment
- Repair locally
- **Min-conflicts:** Select new values that minimally conflict with the other variables
  - Use in conjunction with hill climbing or simulated annealing or…
- **Local maxima strategies**
  - Random restart
  - Random walk

85

Intelligent Backtracking

- **Backjumping**: if V_j fails, jump back to the variable V_i with greatest i such that the constraint (V_i, V_j) fails (i.e., most recently instantiated variable in conflict with V_j)
- **Backchecking**: keep track of incompatible value assignments computed during backjumping
- **Backmarking**: keep track of which variables led to the incompatible variable assignments for improved backchecking

86

Challenges

- What if not all constraints can be satisfied?
  - Hard vs. soft constraints
  - Degree of constraint satisfaction
  - Cost of violating constraints
- What if constraints are of different forms?
  - Symbolic constraints
  - Numerical constraints [constraint solving]
  - Temporal constraints
  - Mixed constraints

87

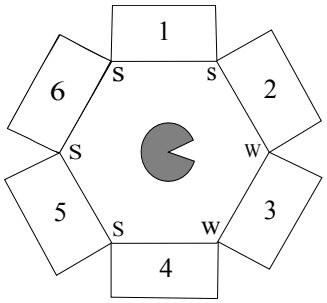
More Challenges

- What if constraints are represented intensionally?
  - Cost of evaluating constraints (time, memory, resources)
- What if constraints/variables/values change over time?
  - Dynamic constraint networks
  - Temporal constraint networks
  - Constraint repair
- What if you have multiple agents or systems involved?
  - Distributed CSPs
  - Localization techniques

88

Trapped!

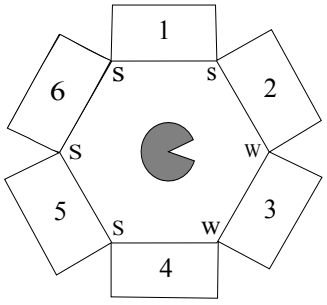
- Pacman is trapped! He is surrounded by mysterious corridors, each of which leads to either a pit (P), a ghost(G), or an exit (E). In order to escape, he needs to figure out which corridors, if any, lead to an exit and freedom, rather than the certain doom of a pit or a ghost.
- While the total number of exits might be zero, one, or more, Pacman knows that two neighboring squares will not both be exits.



89

Trapped!

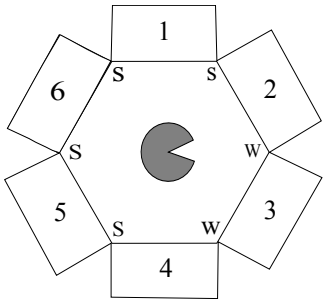
- The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce a breeze.
- Unfortunately, Pacman cannot measure the strength of the breeze at a specific corridor. Instead, he can stand between two adjacent corridors and feel the max of the two breezes.
- For example, if he stands between a pit and an exit he will sense a strong (S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze.



90

Trapped!

- The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce a breeze.
- Unfortunately, Pacman cannot measure the strength of the breeze at a specific corridor. Instead, he can stand between two adjacent corridors and feel the max of the two breezes.
- For example, if he stands between a pit and an exit he will sense a strong (S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze.

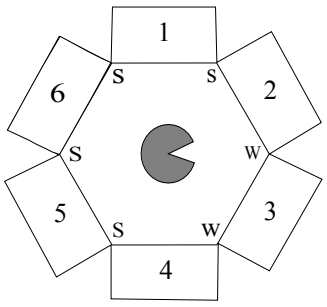


Variables?

91

Trapped!

- The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce a breeze.
- Unfortunately, Pacman cannot measure the strength of the breeze at a specific corridor. Instead, he can stand between two adjacent corridors and feel the max of the two breezes.
- For example, if he stands between a pit and an exit he will sense a strong (S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze.

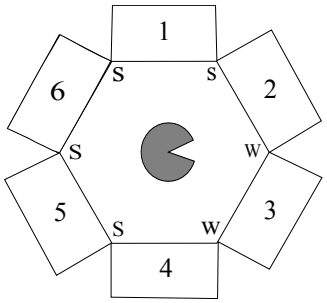


Variables: $X_1 .. X_6$
Domains?

92

Trapped!

- The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce a breeze.
- Unfortunately, Pacman cannot measure the strength of the breeze at a specific corridor. Instead, he can stand between two adjacent corridors and feel the max of the two breezes.
- For example, if he stands between a pit and an exit he will sense a strong (S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze.

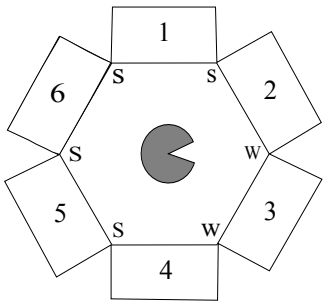


Variables: $X_1 .. X_6$
Domains: {P, G, E}

93

Trapped!

- A pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce a breeze.
- Pacman feels the max of the two breezes.
- The total number of exits might be 0–6.
- Two neighboring squares will not both be exits.

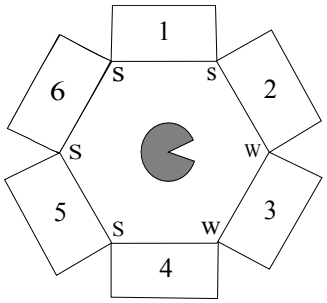


Variables: $X_1 .. X_6$
Domains: {P, G, E}
Constraints?

94

Trapped!

- A pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce a breeze.
- Pacman feels the max of the two breezes.
- The total number of exits might be 0–6.
- Two neighboring squares will not both be exits.



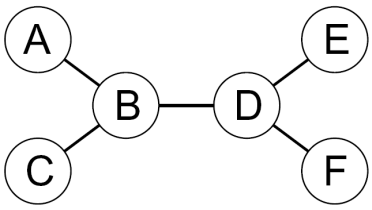
$X_1 = P$ or $X_2 = P$
$X_2 = E$ or $X_3 = E$
$X_3 = E$ or $X_4 = E$
$X_4 = P$ or $X_5 = P$
$X_5 = P$ or $X_6 = P$
$X_6 = P$ or $X_1 = P$

$X_i = E$ nand $X_{i+1|7} = E$

Also! $X_2 =/= P$
$X_3 =/= P$
$X_4 =/= P$

Variables: $X_1 .. X_6$
Domains: {P, G, E}
Constraints?

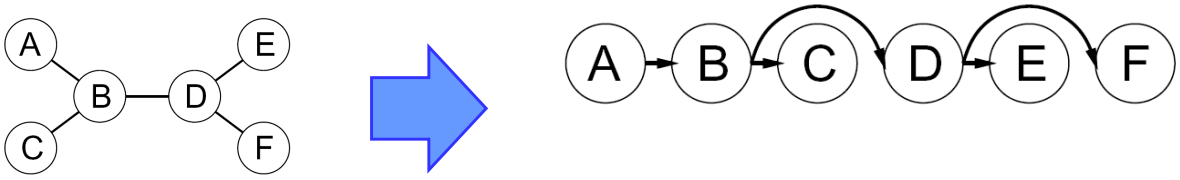
Tree-Structured CSPs



- If the constraint graph has no loops, CSP can be solved in $O(n*d^2)$ time
  - In general CSPs, worst-case time is $O(d^n)$
- This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning

Tree-Structured CSPs

- Algorithm for tree-structured CSPs:
  - Order: Choose a root variable, order variables so that **parents precede children**
  - Remove backward: For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
  - Assign forward: For i = 1 : n, assign X_i consistently with Parent(X_i)

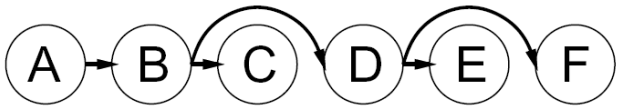


- Runtime: $O(nd^2)$

97

Tree-Structured CSPs

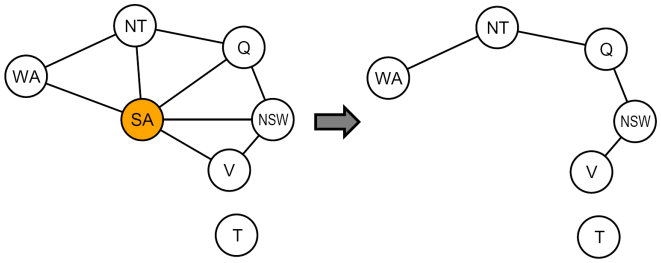
- Claim 1: After backward pass, all root-to-leaf arcs are consistent
  - Proof: Each X→Y was made consistent at one point and Y's domain could not have been reduced thereafter (because Y's children were processed before Y)



- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
  - Proof: Induction on position
  - Why doesn't this algorithm work with cycles in the constraint graph?
  - Note: we'll see this basic idea again with Bayes' nets

98

Nearly Tree-Structured CSPs



- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime $O((d^c)\ (n\text{-}c)\ d^2)$
  - Very fast for small c

99

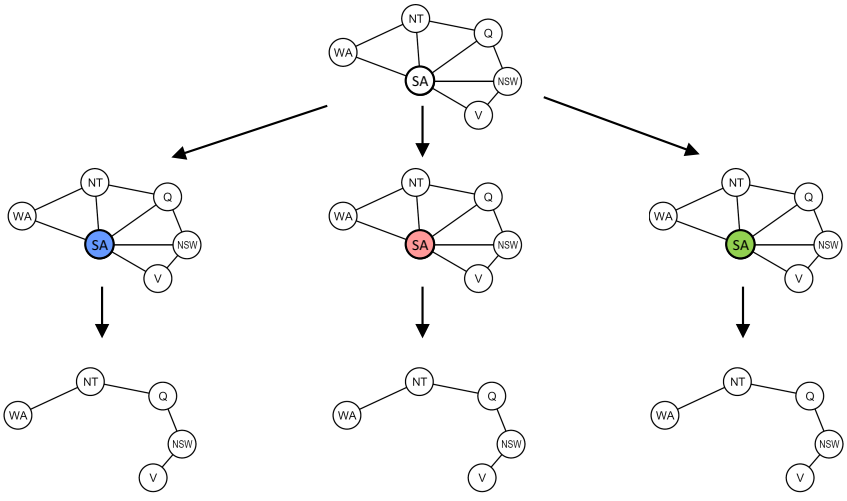
Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

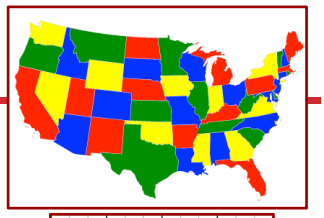
Compute residual CSP for each assignment

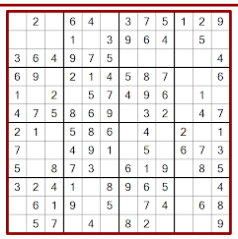
Solve the residual CSPs (tree structured)

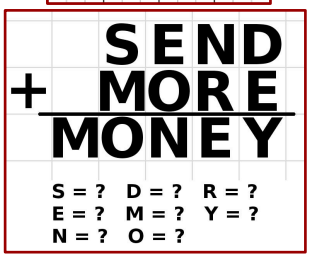


100

CSPs: Summary (1)





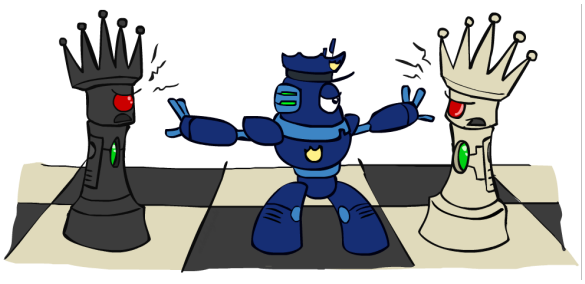


- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by constraints
  - Don't care about path to solution
- **Many problems can be represented as CSPs:** assign variables some value from a domain, then represent constraints among them
- Basic solution: backtracking search
- Speed-ups:
  - Ordering, filtering, structure (cutset conditioning)

Images: docs.ocean.dwavesys.com/en/latest/examples/map_kerberos.html, sudoku-puzzles.net/butterfly-sudoku-easy, medium.com/@co.2020.satenpe/cryptarithmetic-puzzles-are-a-captivating-fusion-of-mathematics-logic-and-wordplay-that-79e3eb619832

101

CSPs: Summary (2)



- CSPs can be represented as **constraint networks** that allow for constraint propagation, tree structuring
- Perform constraint propagation to solve simple problems, or…
  - …search through possible assignments of values to variables
  - …considering **most constrained** variables first
  - …considering the **least constrained** values first
- Worst-case is NP-complete, but in practice we can solve quite hard problems!

102