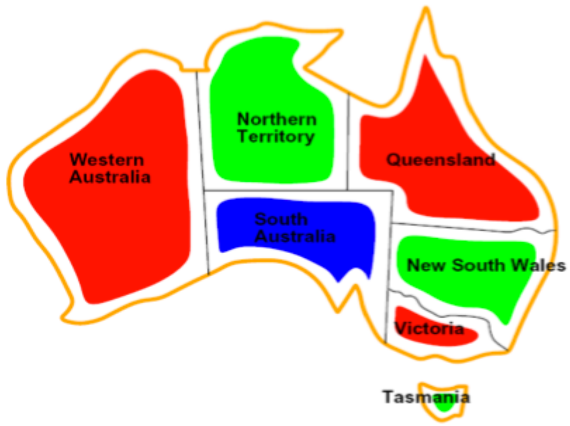
Constraint Satisfaction

Ch. 6.1–6.4 (skip 6.3.3))



Based on slides by: Marie desJardin, Paula Matuszek, Luke Zettlemoyer, Dan Klein, Stuart Russell, Andrew Moore

1

Bookkeeping

- Homework 2 out tonight by midnight (on the schedule)
- Last time: local search, intro to constraint satisfaction problems
- Today: solving CSPs

2

Bookkeeping

- Homework 2 out tonight by midnight (on the schedule)
- Last time: local search, intro to constraint satisfaction problems
- Today: solving CSPs

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*

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.

5

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

6

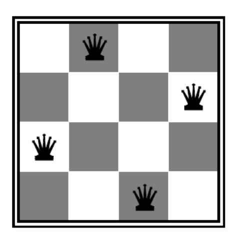
Search Vocabulary

- 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

7

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.



8

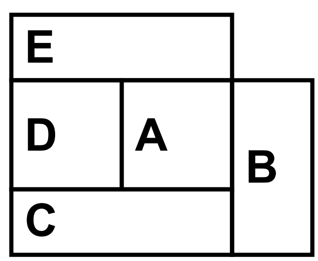
CSP Applications

- 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"?

9

Informal Example: Map Coloring

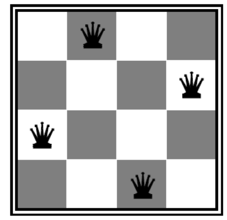
- Given a 2D map, it is always possible to color it using three colors
- Such that:
  - No two adjacent regions are the same color
  - Start thinking: What are the values, variables, constraints?



10

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

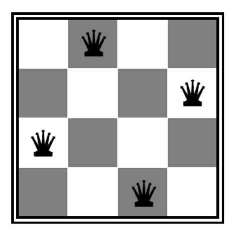


11

Example: N-Queens (1)

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

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

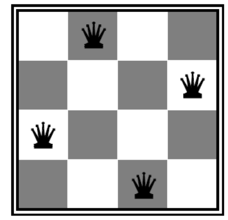


12

Example: N-Queens (2)

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

Q_1
Q_2
Q_3
Q_4



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

-or-

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

• • •

13

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

14

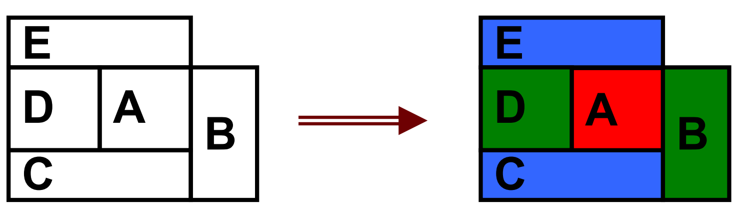
Real-World Problems

- Scheduling
- Temporal reasoning
- Building design
- Planning
- Optimization/satisfaction
- Vision
- Graph layout
- Network management
- Natural language processing
- Molecular biology / genomics
- VLSI design

15

Exercise: Map Coloring II

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



16

Formal Definition: Constraint Network (CN)

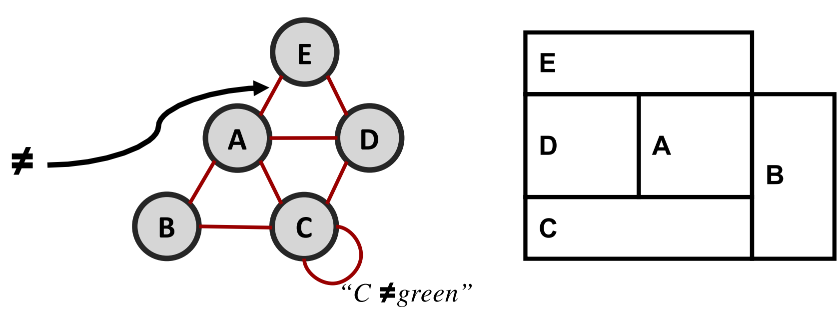
A **constraint network** (CN) consists of

- 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 $R_i \subseteq D_{i1} \times D_{i2} \times \ldots D_{ik}$

17

Constraint Restrictions

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



18

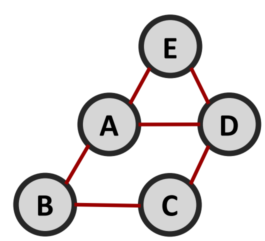
Formal Definition of a CN (cont.)

- 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: $\underline{Q_2 = \{2,3\} \wedge Q_3 = \{1,1\}}$ **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

19

Typical Tasks for CSP

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



20

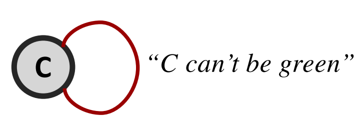
Binary CSP

- **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”

21

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



22

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}

23

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}

24

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)

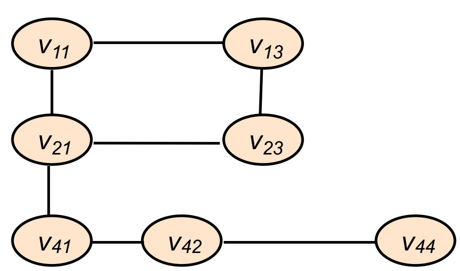
v_{11}	3	v_{13}	1
v_{21}	1	v_{23}	4
3	4	1	2
v_{41}	v_{42}	4	v_{44}

31

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}



32

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

33

Generate and Test: Sudoku

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

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

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

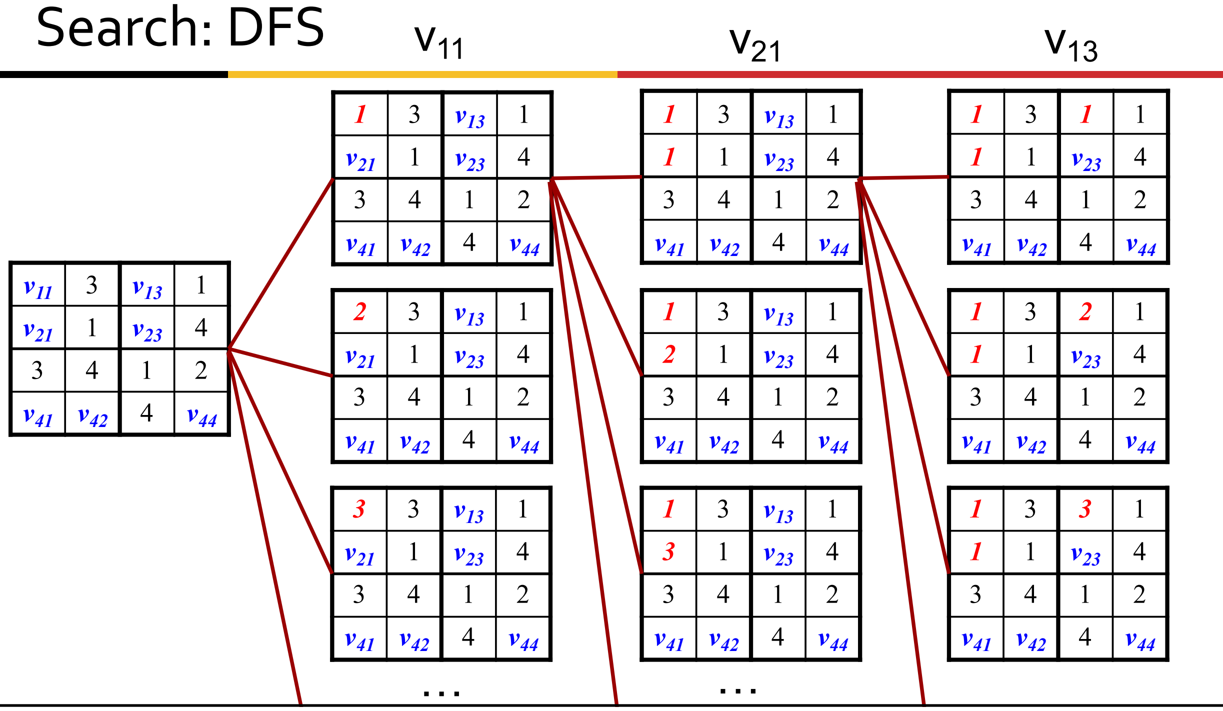
1	3	*1*	1
1	1	*1*	4
3	4	1	2
1	*1*	4	*3*

…

- 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)

34

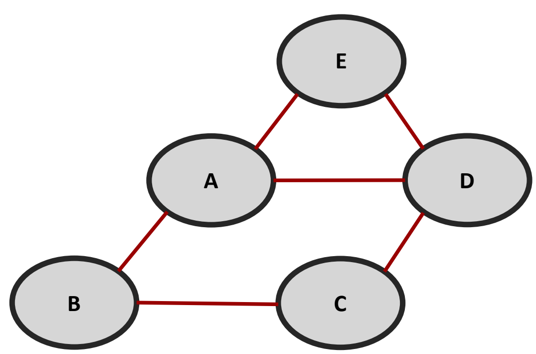
Search: DFS



35

Consistency

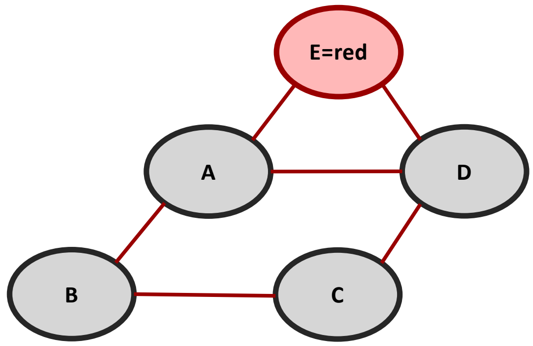
- The goal is to find a solution that is consistent with (doesn't violate) constraints
- An instantiation (assignment of values to variables) is said to be **consistent** if no constraints are violated



36

Consistency

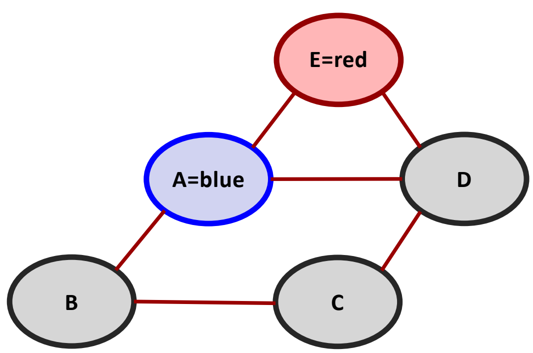
- The goal is to find a solution that is consistent with (doesn't violate) constraints
- An instantiation (assignment of values to variables) is said to be **consistent** if no constraints are violated



37

Consistency

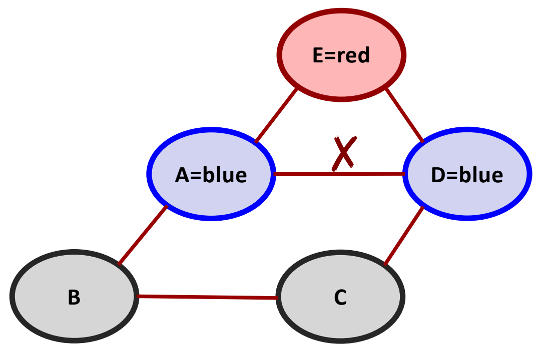
- The goal is to find a solution that is consistent with (doesn't violate) constraints
- An instantiation (assignment of values to variables) is said to be **consistent** if no constraints are violated



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

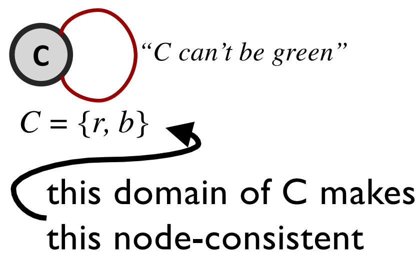
Consistency

- 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

40

Node Consistency

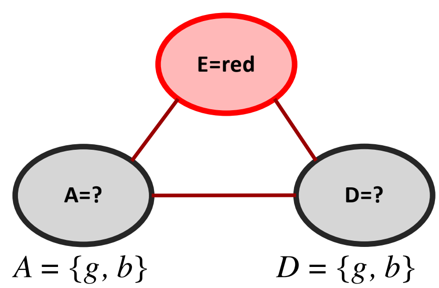
- **Node consistency:** every value in **node X**'s domain (every value we think it might take) is consistent with *X's unary constraints*
  - A graph is node-consistent if all nodes are node-consistent
  - Let's say C can't be green
  - C = {red, ~~green~~, blue}



41

Arc Consistency

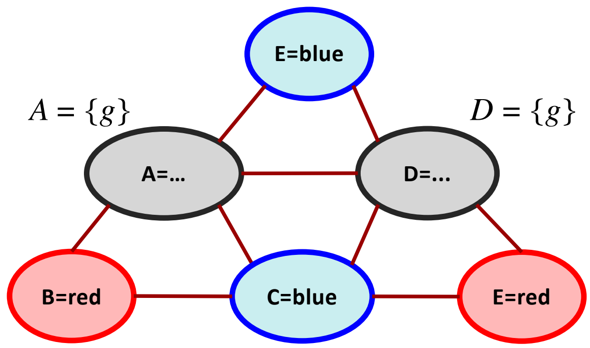
- **Arc consistency:** A variable in a CSP is **arc-consistent** if every value in its domain satisfies the variable's binary constraints
- For every value *x* of X in Arc(X,Y):
  - **∃*y*** for Y
  - That satisfies the constraint represented by the arc
- A graph is arc-consistent if all arcs are arc-consistent



42

Arc Consistency: Example

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

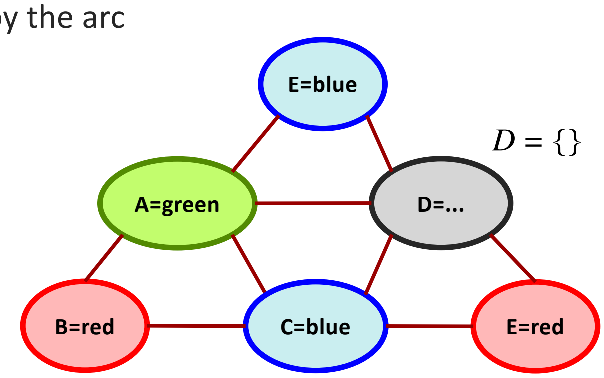


- Is this instantiation arc-consistent?
- So far, yes!

43

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?

44

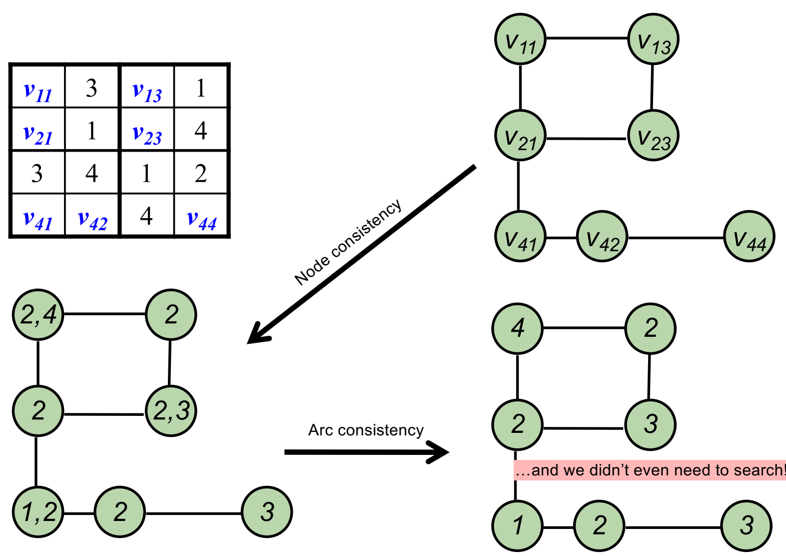
Constraint Propagation

- 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

45

Constraint Propagation: Sudoku

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



46

Example: Map-Coloring

- Variables: WA, NT, Q, NSW, V, SA, T
- Domain: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors
  - Ex: $WA \neq NT$

$$(WA, NT) \in \{(red, green), (red, blue), (green, blue), (green, red), (blue, red)\}$$

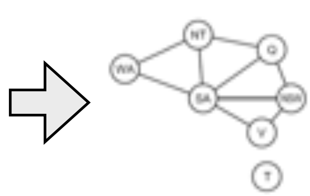
- Solutions are assignments satisfying all constraints, e.g.:

$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$$

47

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!

48

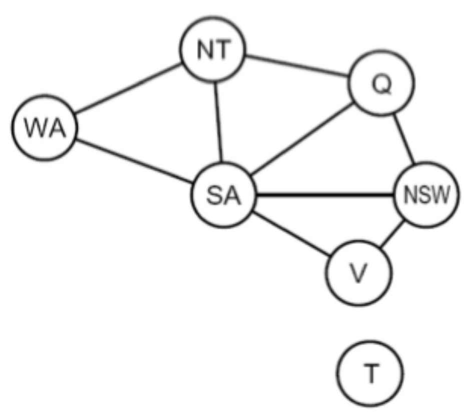
Standard Search Formulation

- 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

49

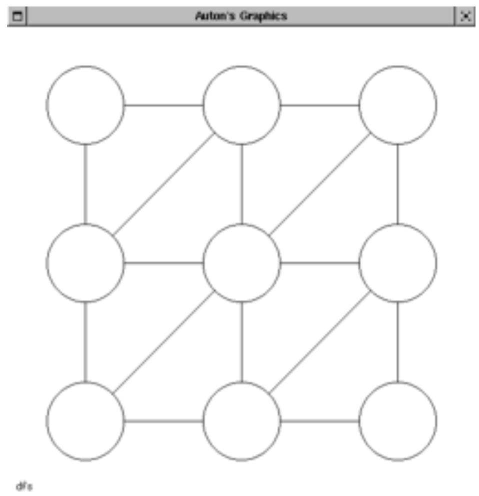
Search Methods

- What does BFS do?
- What does DFS do?



50

DFS: not good!



51

Backtracking Search

- DFS is bad. So how do we improve it?
- **Idea 1: Only consider a single variable at each point**
  - Variable assignments are commutative, so fix the ordering
    - Ex: [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
  - How many leaves are there now?
- **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"

52

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**
- 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

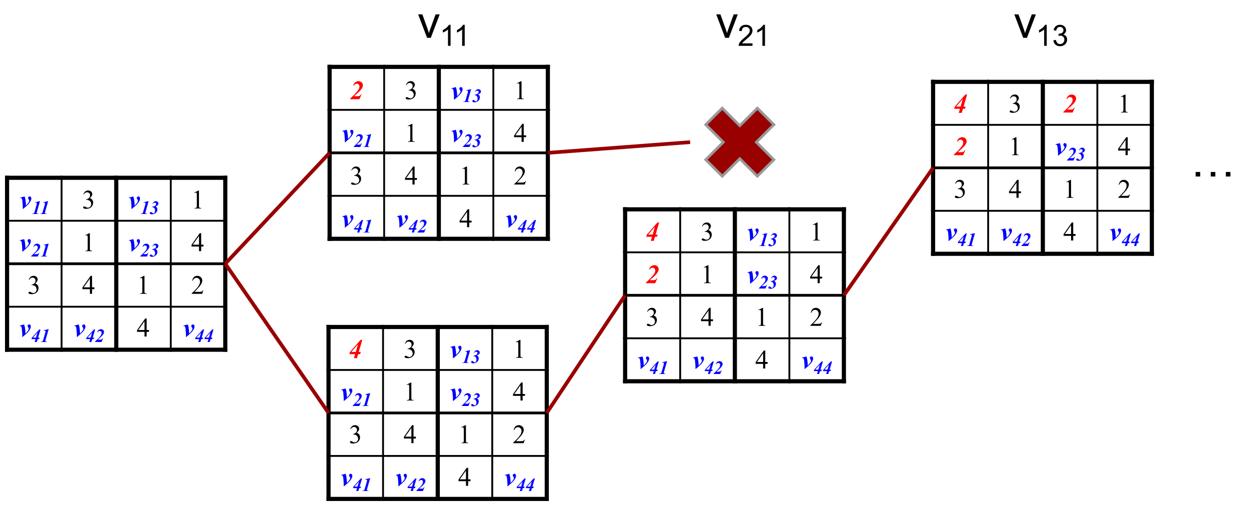
53

Backtracking Search

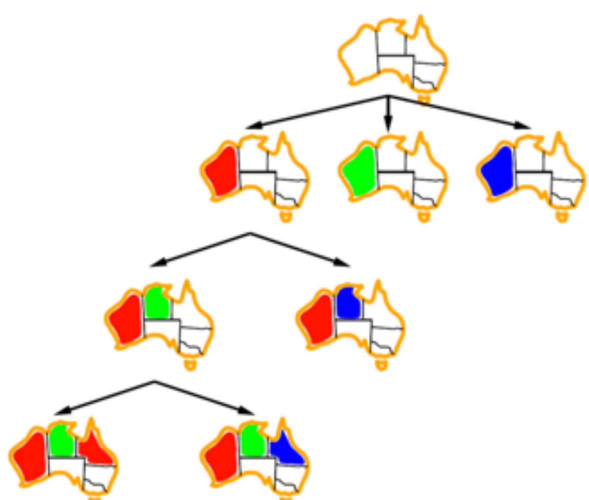
- Idea 1: Only consider a single variable at each point
- Idea 2: Only allow legal assignments at each point
- DFS for CSPs with these two improvements is called **backtracking search**
  - We *backtrack* when there's no legal assignment for the next variable
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for $n \approx 25$

54

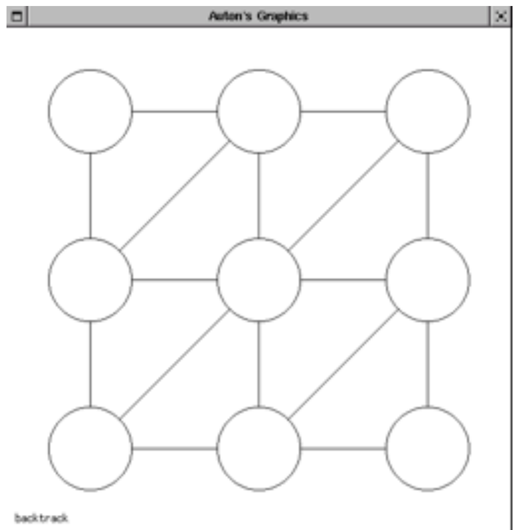
Backtracking: Sudoku



Backtracking Example

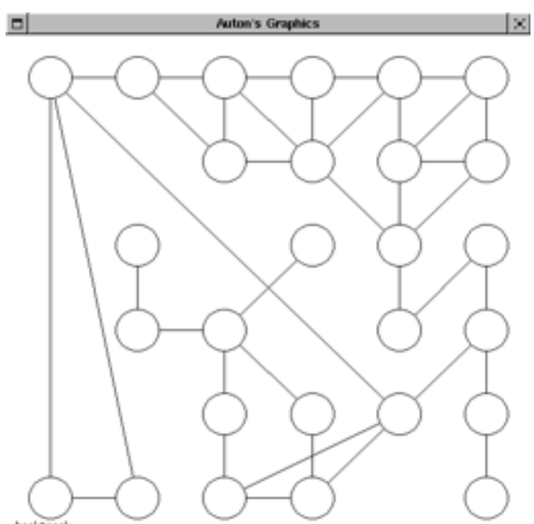


Backtracking



58

Good Enough?



59

Problems with Backtracking

- Thrashing: keep repeating same failed variable assignments
  - Consistency checking can help
  - Intelligent backtracking schemes can also help

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

- 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

60

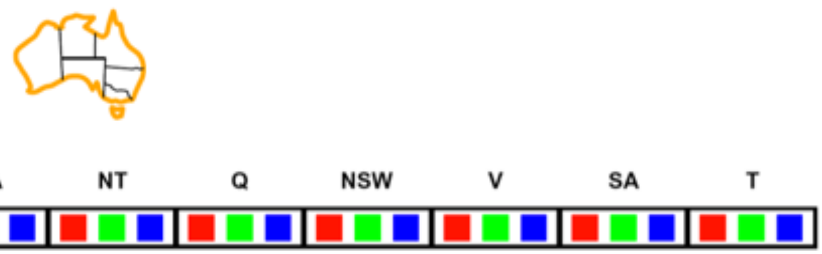
Improving Backtracking

- 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?

61

Forward Checking

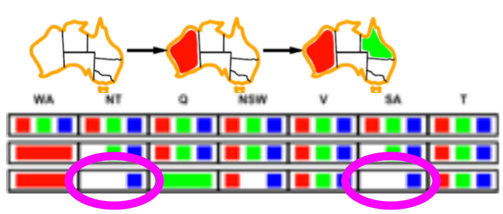
- Idea: Keep track of remaining legal values for unassigned variables (using immediate constraints); terminate when any variable has no legal values



62

Forward Checking

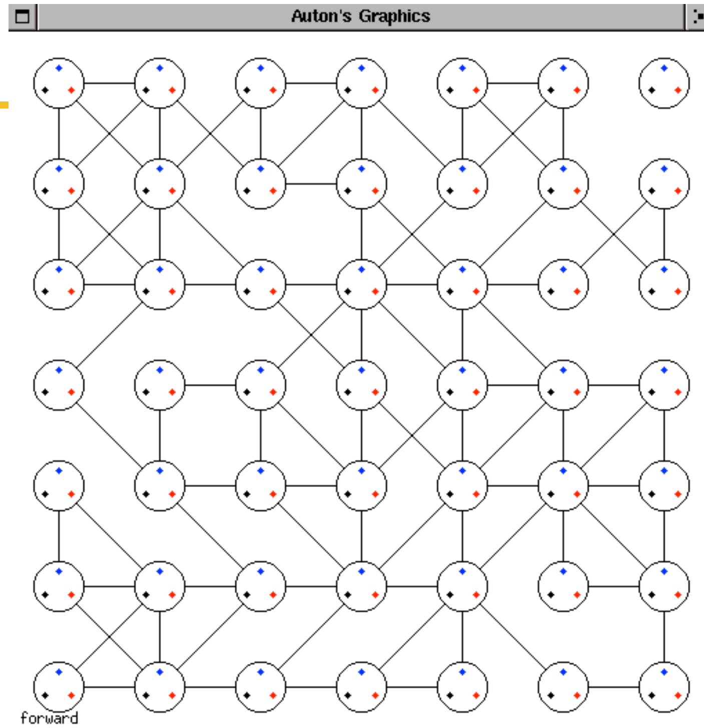
- Propagates information from assigned to **adjacent** unassigned variables
- Doesn’t detect more distant failures



- NT and SA cannot both be blue! Why didn’t we detect this?
- Constraint propagation enforces constraints locally
  - This is a local maximum!

63

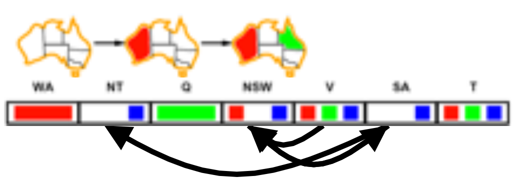
Are We Done?



64

Arc Consistency

- Simplest form of propagation makes each arc consistent
  - X → Y is consistent iff for every value 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
- What's the downside of arc consistency?
- Can be run as a preprocessor or after each assignment

65

K-consistency

- K-consistency generalizes the notion of arc consistency to sets of **more than two variables**
- A graph is **K-consistent** if, for legal values of any K-1 variables in the graph, and for any K^{th} variable V_k, there is a legal value for V_k
- **Strong** K-consistency = J-consistency for all J≤K
- Node consistency = strong 1-consistency
- Arc consistency = strong 2-consistency
- Path consistency = strong 3-consistency

66

Why Do We Care?

- A strongly N-consistent CSP with N variables can be solved **without backtracking**
- For any CSP that is strongly K-consistent:
  - **If** we find an appropriate variable ordering (one with "small enough" branching factor)
  - We can solve the CSP without backtracking

67

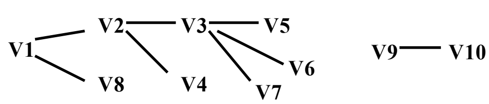
Ordered Constraint Graphs

- Select a variable ordering, $V_1, \ldots, V_n$
- **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)), $1 \le i \le N$
- **Width of an unordered CG** is the minimum width of all orderings of that graph (best you can do)

68

Tree-Structured Constraint Graph

- A constraint tree rooted at V_1 satisfies:
  - There exists an ordering $V_1, \ldots, V_n$ such that **every node has zero or one parents** (i.e., each node only has constraints with at most one "earlier" node in the ordering)



  - Also known as an ordered constraint graph with width 1
- If this constraint tree is also node- and arc-consistent (a.k.a. strongly 2-consistent), it can be solved without backtracking
  - (More generally, if the ordered graph is strongly k-consistent, and has width w < k, then it can be solved without backtracking.)

69

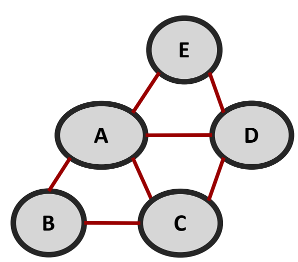
So What If We Don't Have a Tree?

- Answer #1: Try **interleaving** constraint propagation and backtracking
- Answer #2: Try using **variable-ordering** heuristics to improve search
- Answer #3: Try using **value-ordering** heuristics during variable instantiation
- Answer #4: See if **iterative repair** works better
- Answer #5: Try using **intelligent backtracking methods**

72

Possible Variable Orderings

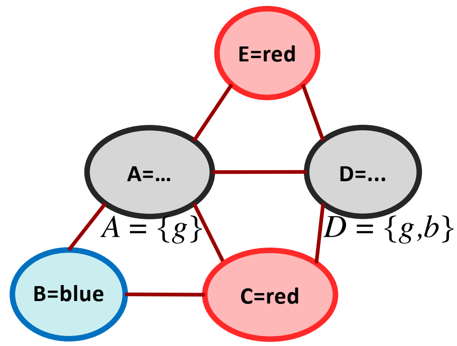
- **Intuition**: choose variables that are highly constrained early in the search process; leave easy ones for later.
- How?
  - **Minimum width ordering** (MWO): identify OCG with minimum width
  - **Maximum cardinality ordering**: approximation of MWO that's cheaper to compute: order variables by decreasing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



74

Possible Variable Orderings

- **Fail first principle** (FFP): choose variable with fewest remaining values
  - AKA **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



75

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

76

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

77

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)

78

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

80

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

81

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

82

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

84

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

85

Summary

- Many problems can be represented as CSPs: assign **variables** some **value from a domain**, then represent **constraints** among them
- 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 CSPs are NP-complete, but in practice we can usually solve quite hard problems!

86