CIS 471/571(Fall 2020): Introduction to Artificial Intelligence

Lecture 4: Co ction Problems (Part

Thanh H. Nguyen

Source: http://ai.berkeley.edu/home.html

Reminder

- Homework 1: Search
  - Deadline: Oct 10th, 2020

- Project 1: Search
  - Deadline: Oct 13th, 2020

Today

- Constraint Satisfaction Problems
  - Backtracking Search
  - Filtering
  - Ordering

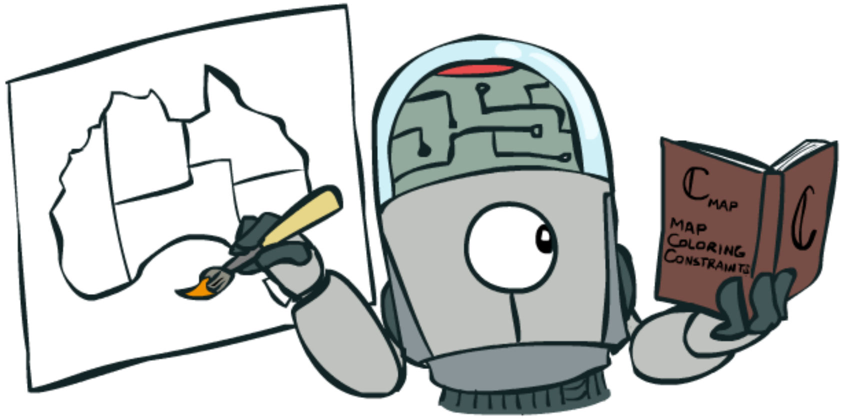
Constraint Satisfaction Problems

Constraint Satisfaction Problems

- 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 problem
  - A special subset of search problems

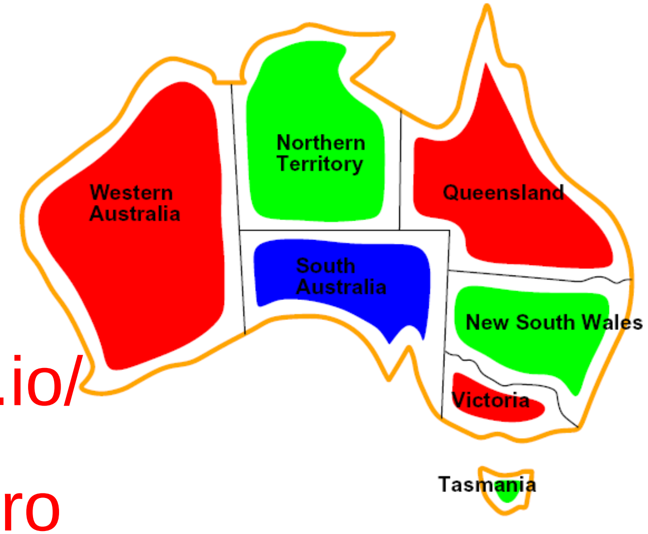
  - State is defined by variables X_i with values fr domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables



- Allows useful general-purpose algorithms with more power than standard search algorithms

CSP Examples

Example: Map Coloring



- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D = \{\text{red}, \text{green}, \text{blue}\}$

- Constraints: adjacent regi

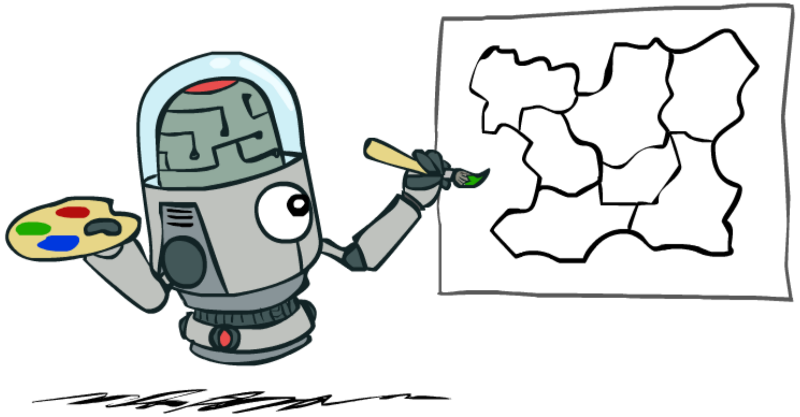
different colors

Implicit: $WA \neq NT$

Explicit: $(WA, NT) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), \ldots\}$

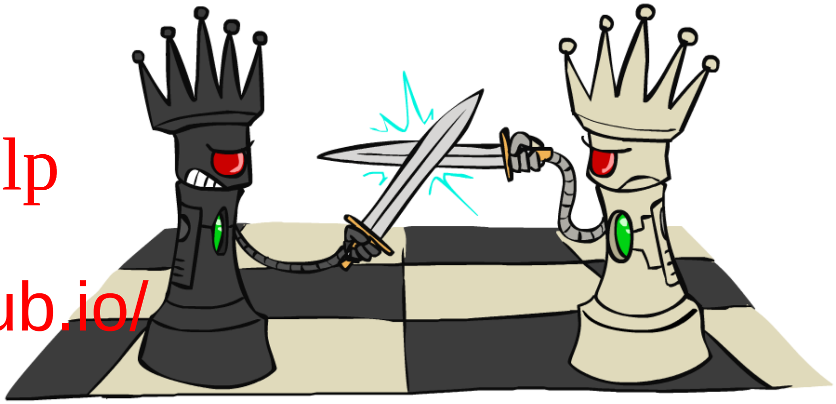
- Solutions are assignments satisfying all constraints, e.g.:

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



Example: N-Queens

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



$$\forall i, j, k \ \ (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

Example: N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains:
  - Constraints:

Q_1

Q_2

Q_3

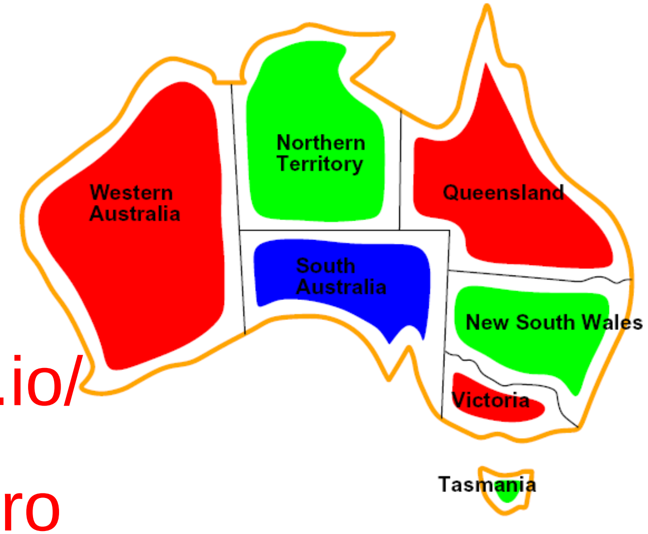
Q_4

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

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

$\ldots$

Constraint Graphs

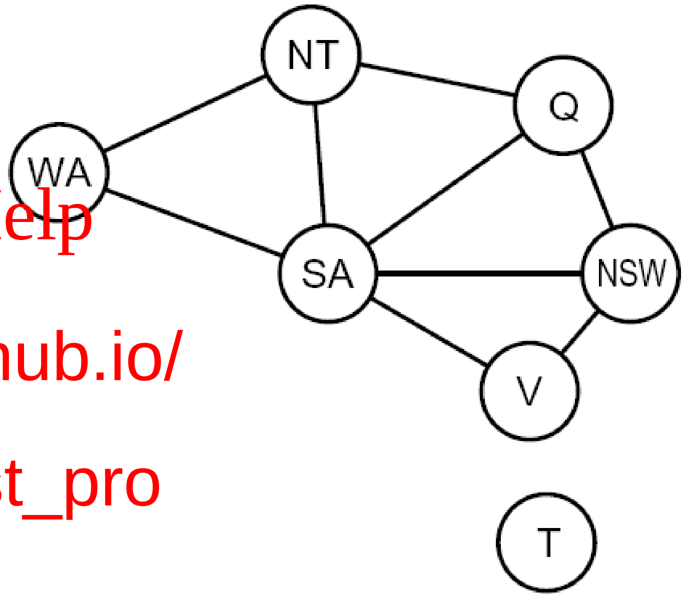


Constraint Graphs

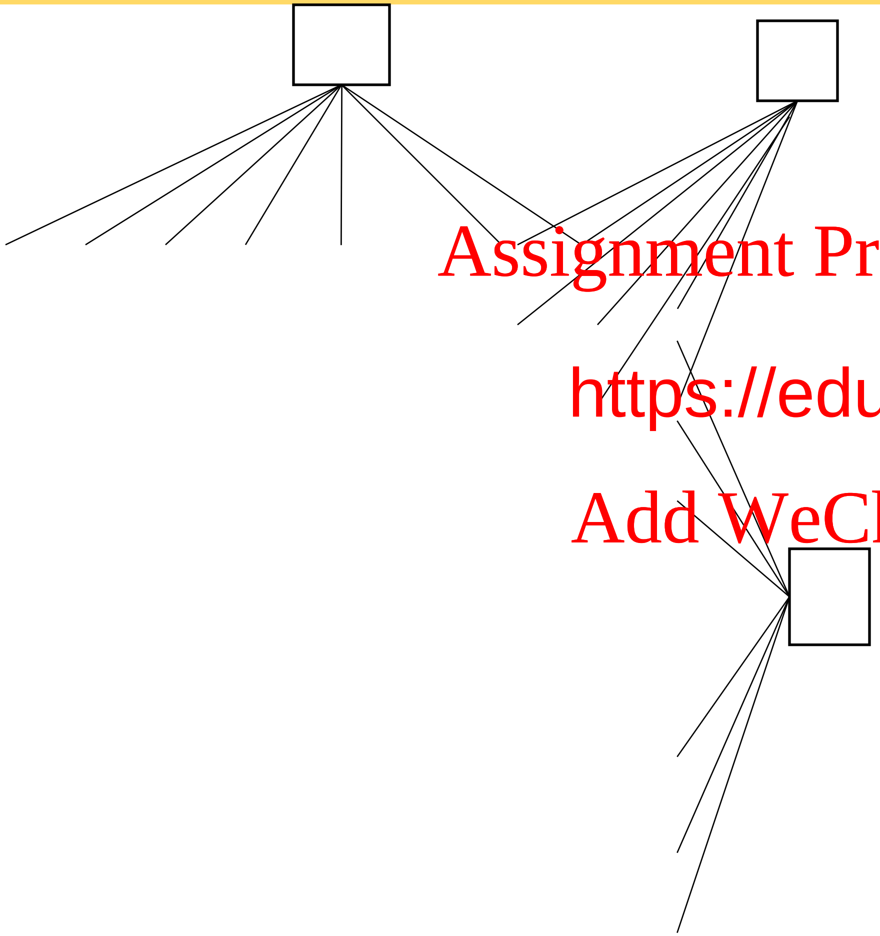
- Binary CSP: each constraint relates (at most) two variables

- Binary constraint graph: arcs show constraints

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



Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - ,2,…,9}
- traints:
  - y 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)

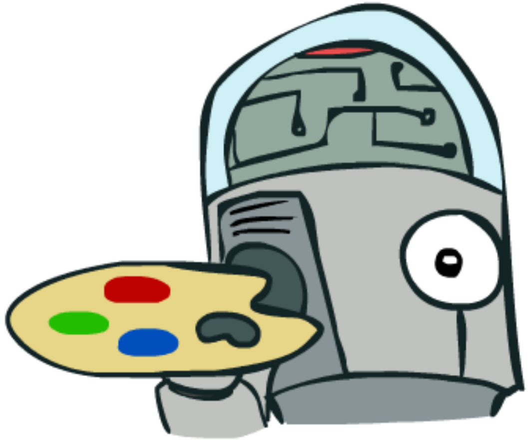
Varieties of CSPs and Constraints

Varieties of CSPs

- Discrete Variables
  - Finite domains
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)

  - Infinite domains (integers, st

    - E.g., job scheduling, variables are start/end h job



- Continuous variables
  - E.g., start/end times for Hubble Telescope observations

Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equivalent to reducing domains), e.g.: $SA \neq green$
  - Binary constraints invol .g.: $SA \neq WA$
  - Higher-order constraint iables

- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems

Real-World CSPs

- Scheduling problems: e.g., when can we all meet?
- Timetabling problems: e.g., which class is offered when and where?
- Assignment problems: e.g., who teaches what class
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!
- Many real-world problems involve real-valued variables…

Solving CSPs

Standard Search Formulation

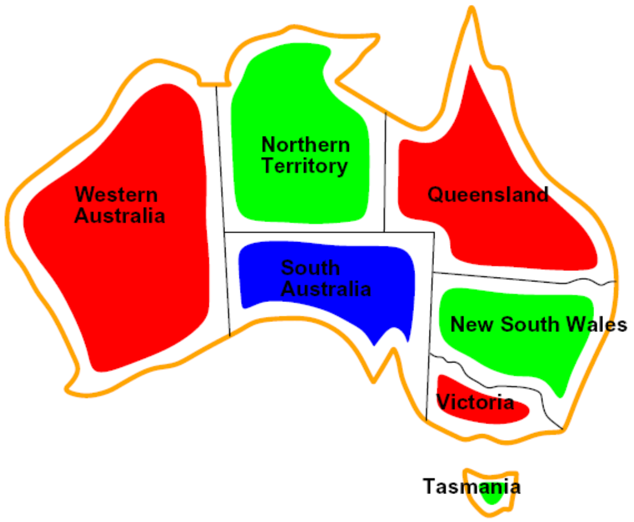
- Standard search formulation of CSPs

- States defined by the values assigned so far (partial assignme
  - Initial state: the empty ass
  - Successor function: assign unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

- We'll start with the straightforward, naïve approach, then improve it

Search Methods

- What would BFS do?



- What would DFS do?

- What problems does naïve search have?

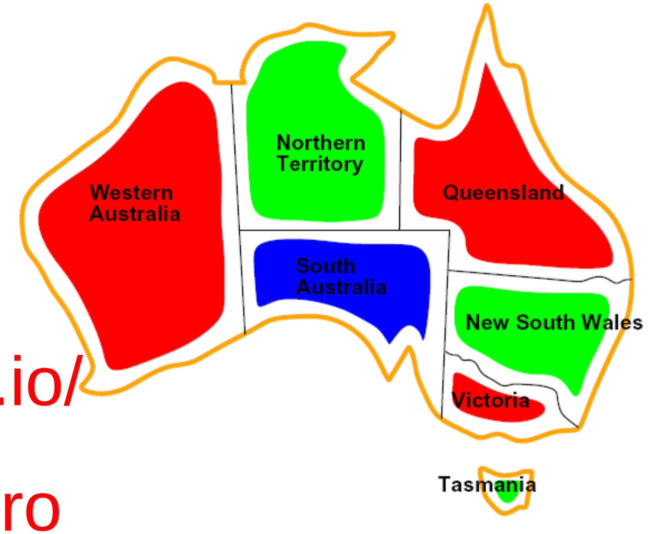
[Demo: coloring -- dfs]

Backtracking Search

Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- **Idea 1: One variable at a time**
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignm ach step
- **Idea 2: Check constraints as you go**
  - I.e. consider only values which do not conflict wi ssignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"
- Depth-first search with these two improvements is called *backtracking search* (not the best name)
- Can solve n-queens for $n \approx 25$

Backtracking Example



Backtracking Search

- Backtracking = DFS + variable-ordering + fail-on-violation
- What are the choice points?

Improving Backtracking

- General-purpose ideas give huge gains in speed

- Ordering:
  - Which variable should b
  - In what order should its

- Filtering: Can we detect inevitable failure early?

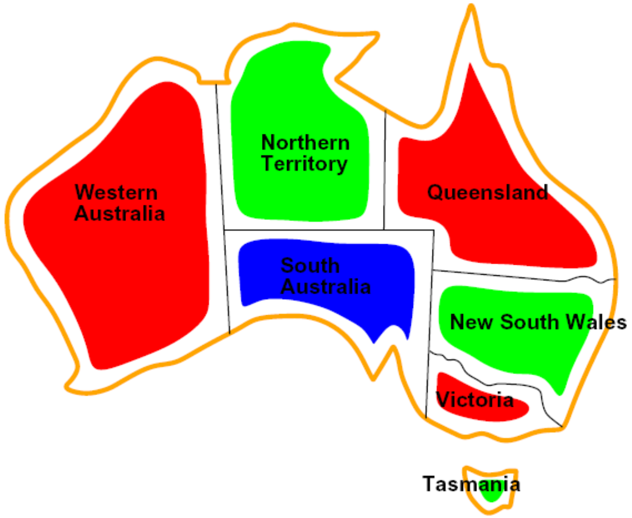
- Structure: Can we exploit the problem structure?

Filtering

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

WA NT Q SA NSW V



Filtering: Constraint Propagation

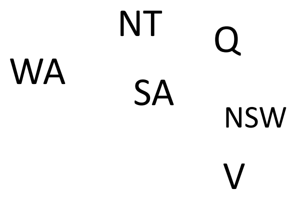
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

NT Q
WA SA
NSW
V

- NT and SA cannot both be blue!
- Why didn’t we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

Consistency of A Single Arc

- 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



Delete from the tail!

- Forward checking: Enforcing consistency of arcs pointing to each new assignment

Arc Consistency of an Entire CSP

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

NT Q
WA SA
NSW
V

- Important: 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
- What's the downside of enforcing arc consistency?

Remember: Delete from the tail!

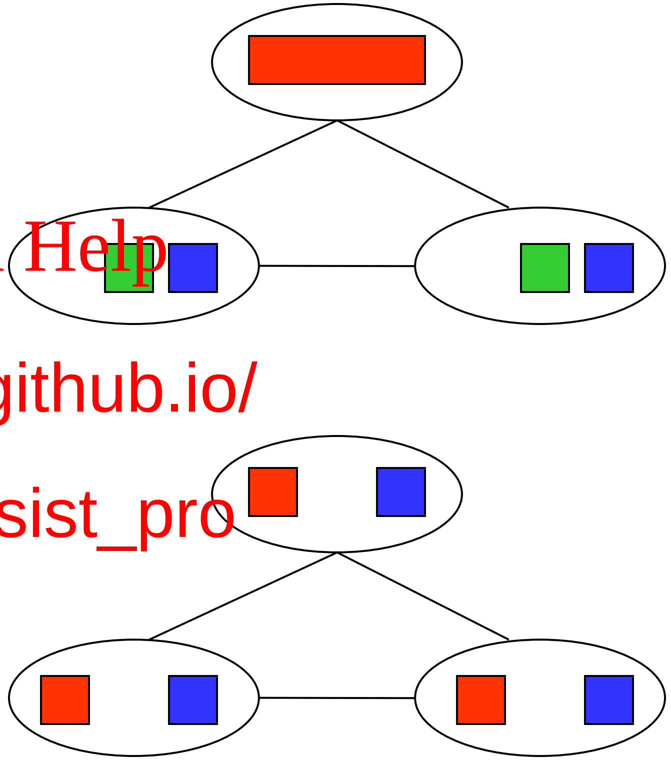
Enforcing Arc Consistency in a CSP

- Runtime: $O(n^2d^3)$

Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple s
  - Can have no solution
    not know it)

- Arc consistency still runs inside a backtracking search!

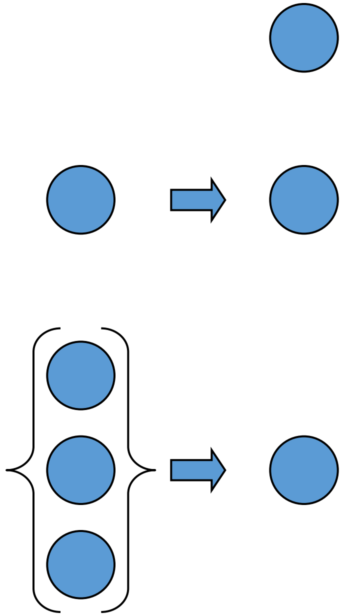


What went wrong here?

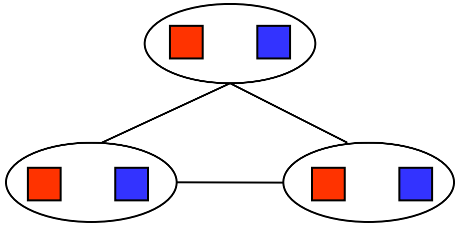
K-Consistency

K-Consistency

- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
  - 2-Consistency (Arc Consisten any consistent assignment to one er
  - K-Consistency: For each k nodes, any consiste nt to k-1 can be extended to the k^{th} node.



- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)



Strong K-Consistency

- 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 an

  - Choose a new variable
  - By 2-consistency, there is a choice consistent

  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - …

- Lots of middle ground between arc consistency and n-consistency! (e.g. k=3, called path consistency)