Constraint Satisfaction Problems

Lecture 5, CMSC 170

John Roy Daradal / Instructor

Previously on CMSC 170

- Search and Planning
- Uninformed Search (BFS, DFS, UCS)
- Informed Search (Greedy, A*)
- Heuristics

Today's Topics

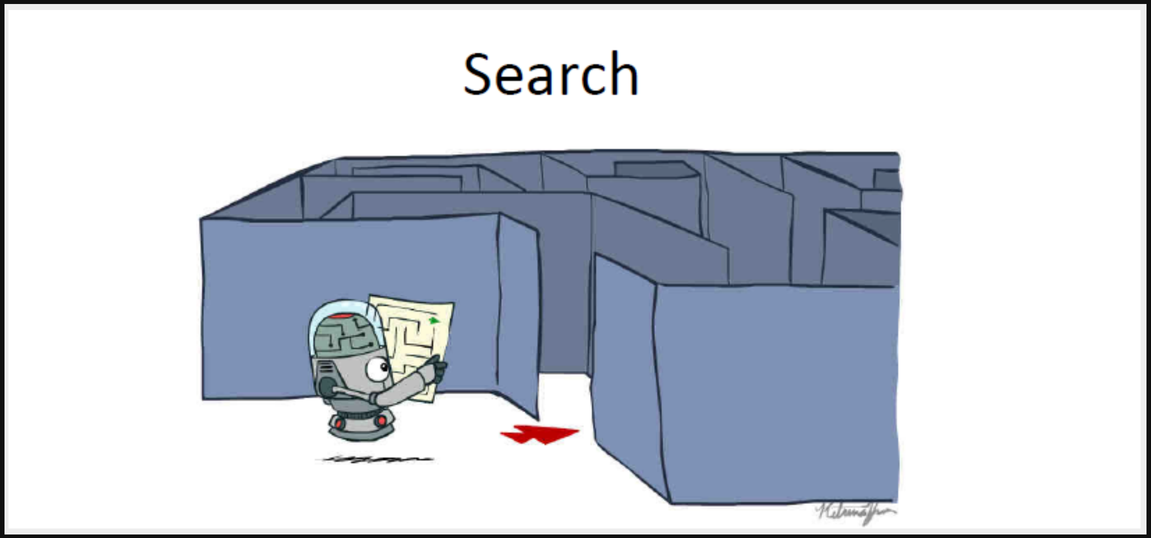
- Constraint Satisfaction Problems
- CSP Modeling
  - Variables
  - Domains
  - Constraints
- Backtracking Search

Today's Topics

Improving Backtracking:

- **Filtering**: forward checking
- **Ordering**: variable, value
- **Avoiding thrashing**: backjumping, nogoods recording

Search



Search

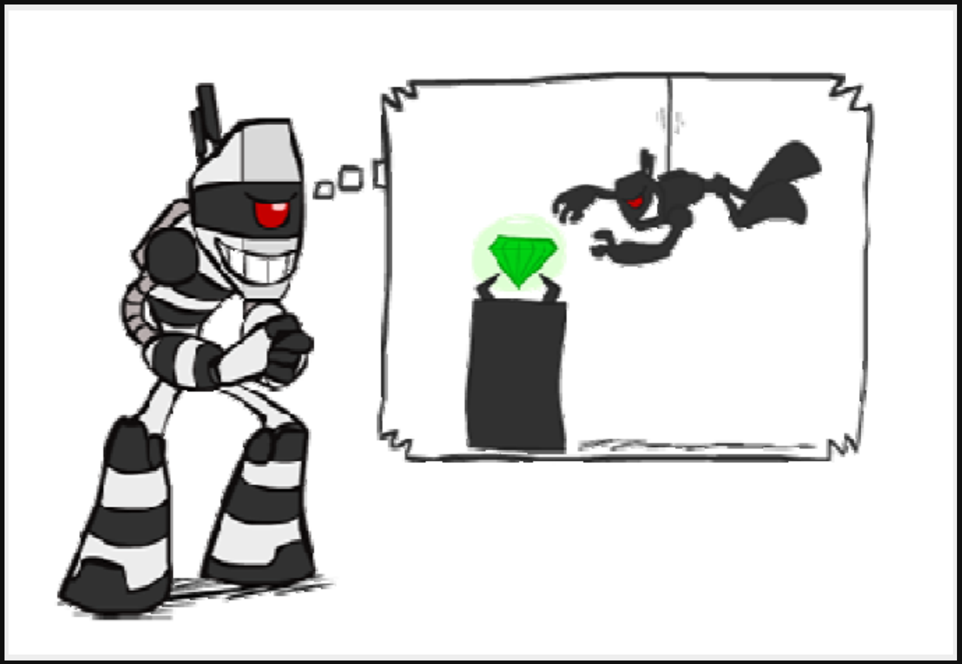
Environment

- **Single** agent
- **Deterministic** actions
- **Fully** observable
- **Discrete** state space

Search

Planning vs Identification

Planning



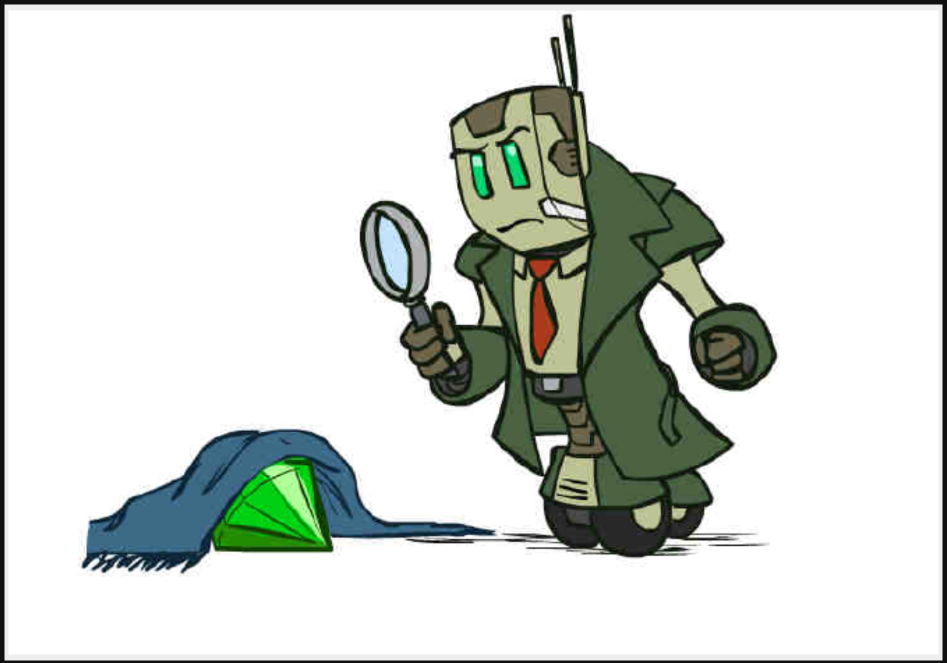
Planning

- *Output*: sequence of **actions**
- **Path to goal** is important
- *Example*: getting out of maze

Planning

- Paths have various **costs**, **depths**
- **Heuristics** give problem-specific *guidance* (to solve *faster*)

Identification



Identification

- *Output*: **assignments** to variables
- **Goal** itself is important, not the path
- *Example*: constraint satisfaction problems

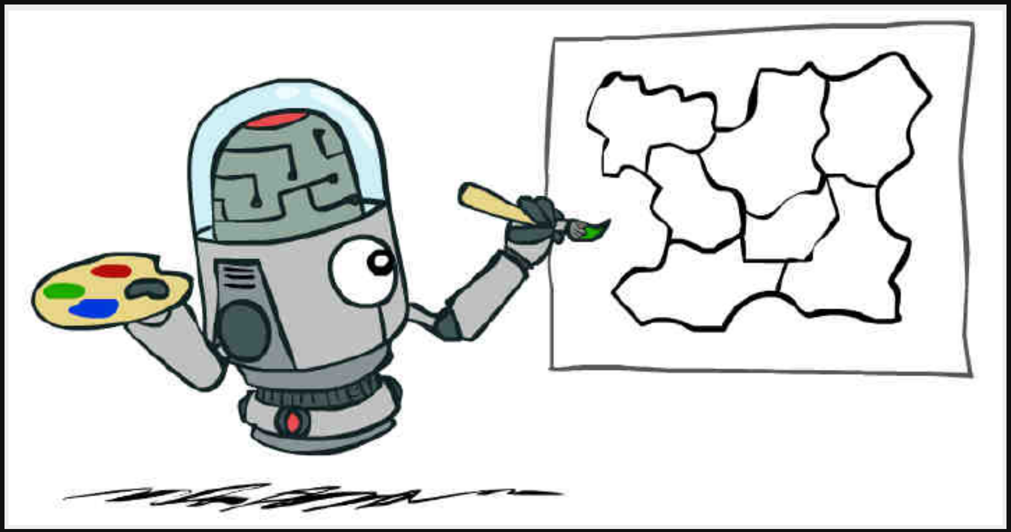
Constraint Satisfaction Problem

- Assign **values** to **variables**, subject to **constraints**
- **Domain**: *values* allowed to be assigned to a variable

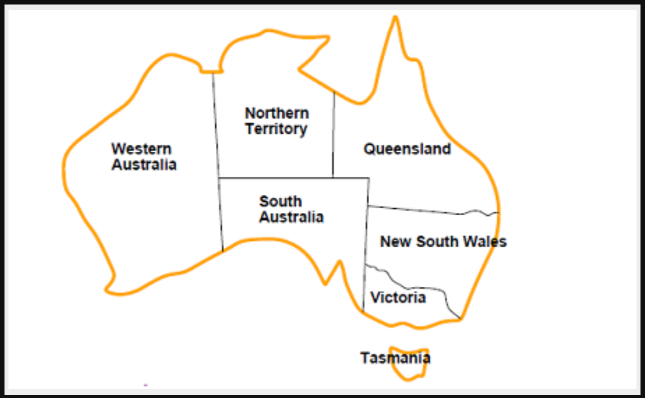
Constraint Satisfaction Problem

- **Constraints**: specify allowable *combinations* of values for subsets of variables
- **Solution**: all variables are *assigned* values from respective domains
- **Correct Solution**: all constraints must be **satisfied**

Map Coloring



Map Coloring

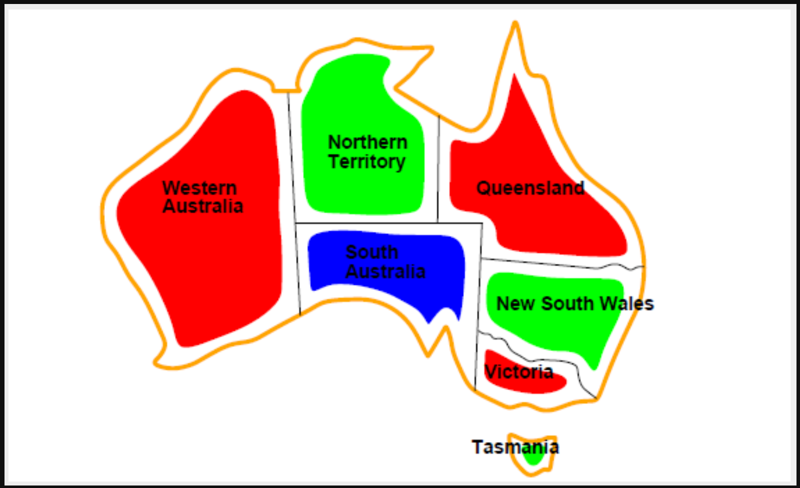


- **Variables**: WA, NT, SA, Q, NSW, V, T
- **Domains**: {**red**, **green**, **blue**}
- **Constraints**: adjacent regions must have different colors

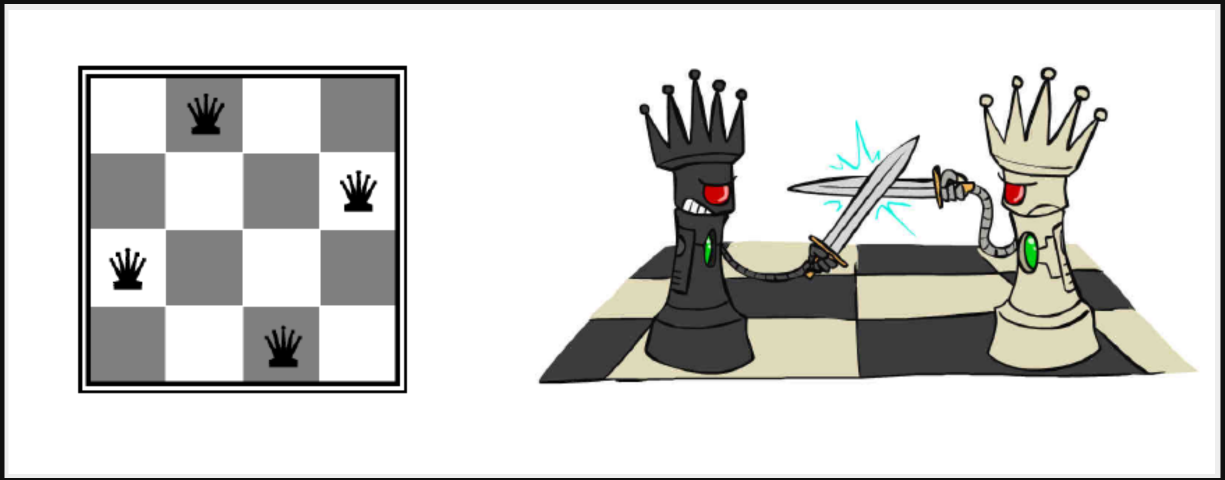
Map Coloring

One solution:

WA = r, NT = g, SA = b, Q = r,
NSW = g, V = r, T = g



N-Queens



- Arrange **N queens** in a **NxN grid**
- No queens **attack** each other
- Horizontal, Vertical, Diagonal

N-Queens

Formulation 1:

- **Variables**: X_{ij} (grid cell)
- **Domains**: {0,1} (queen)

N-Queens

Formulation 1 Constraints:

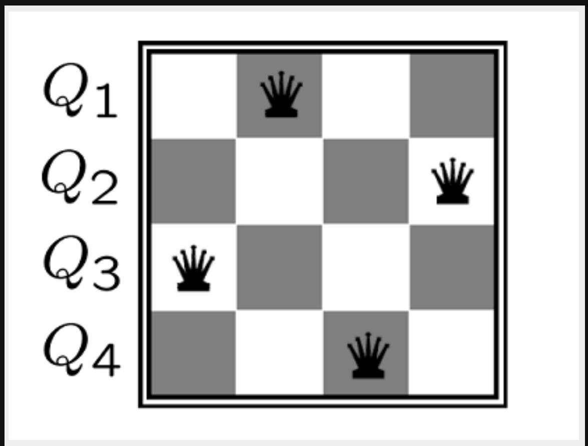
$$\forall i, j, k \;\; (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

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

Example of **explicit** constraints

N-Queens

Formulation 2:



- **Variables**: Q_k (row)
- **Domains**: {1,2,,...,N} (column)

N-Queens

Formulation 2 Constraints:

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

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

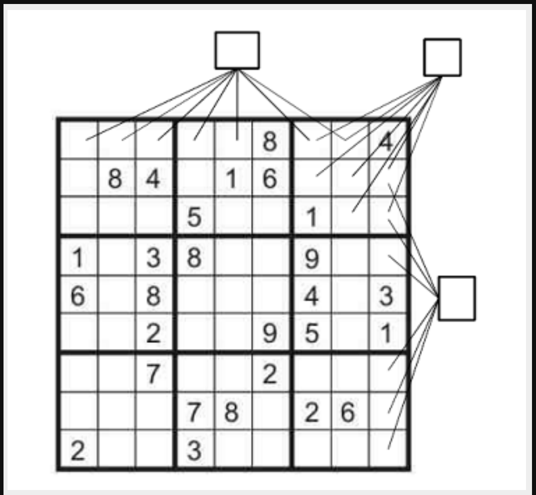
$\cdots$

N-Queens

Which formulation is **better**?

- **Formulation 2** is better - it *encodes* problem knowledge (one per row)
- Formulation 1 enforces row constraint using constraints (not baked into problem)

Sudoku



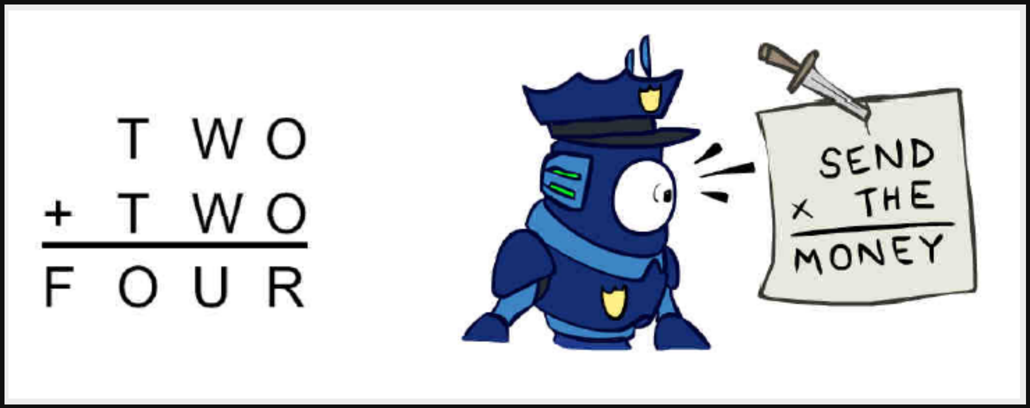
- **Variables**: Each open square
- **Domains**: {1,2,3,4,5,6,7,8,9}

Sudoku

Constraints:

- AllDifferent constraint for each column
- AllDifferent constraint for each row
- AllDifferent constraint for each region

Cryptarithmetic



- **Variables**: F, T, U, W, R, O, X_1, X_2, X_3
- **Domains**: {0,1,2,3,4,5,6,7,8,9}

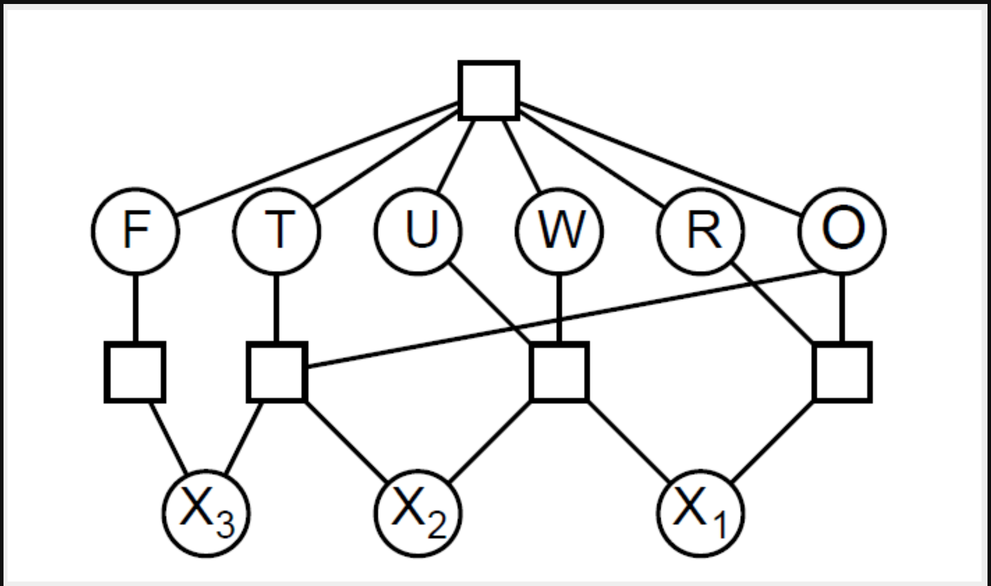
Cryptarithmetic

Constraints (TWO + TWO = FOUR):

- AllDifferent(F, T, U, W, R, O)
- $O + O = R + 10 * X_1$
- $X_1 + W + W = U + 10 * X_2$
- $X_2 + T + T = O + 10 * X_3$
- $X_3 = F$

Cryptarithmetic

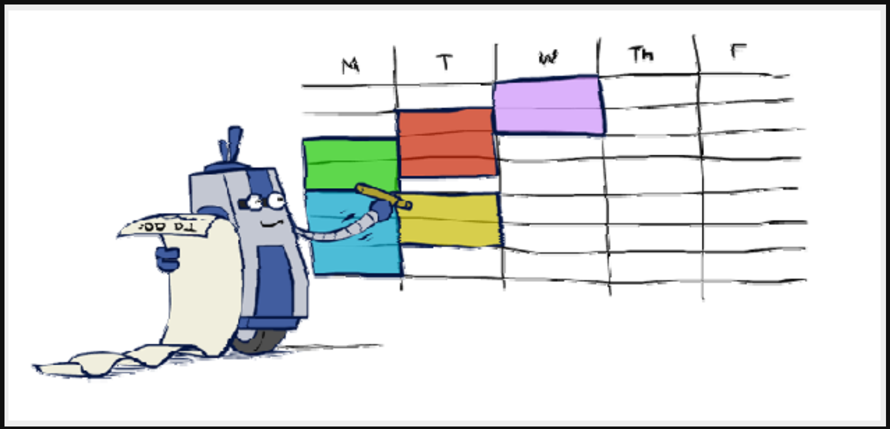
Constraint Graph:



Real-World CSP

- **Assignment**: who teaches a class?
- **Timetabling**: which class is offered when and where (room and timeslot) ?

Real-World CSP



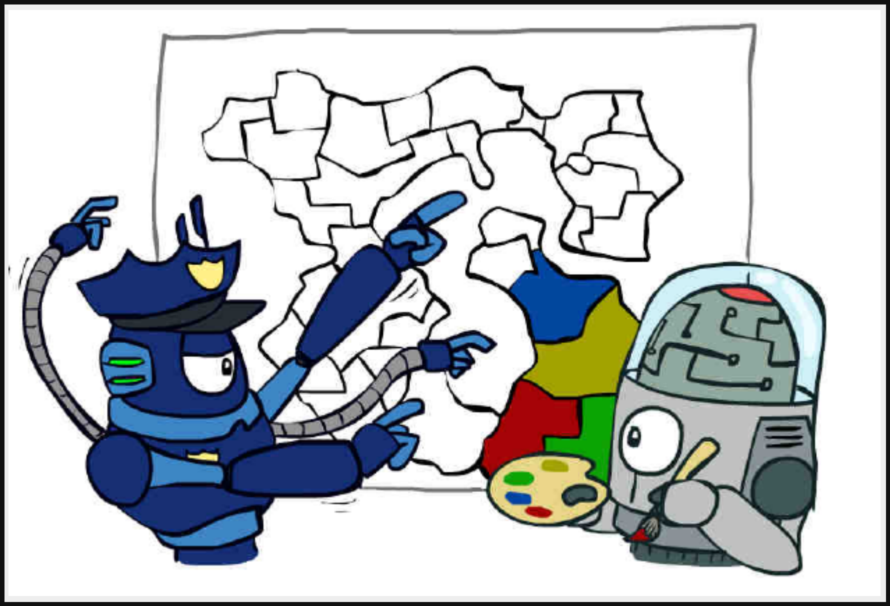
Real-World CSP

- Transportation scheduling
- Factory scheduling
- Sports scheduling
- Floor planning

Real-Valued Variables

- *Focus*: CSP with **discrete** variables
- Many real-world problems involve **real-valued variables**
- Use techniques like **Linear Programming**, or make variables *discrete* before solving

Types of Constraints



Types of Constraints

- Unary / Domain
- Binary
- Global
- Soft

Types of Constraints

Unary Constraint

- aka *domain* constraint
- **single** variable
- reduce domain size
- *example*: SA ≠ green

Types of Constraints

Binary Constraint

- **two** variables
- *example*: SA ≠ WA

Global Constraint

- aka *higher-order* constraint
- **3 or more** variables
- *example*: AllDifferent

Types of Constraints

Hard Constraint

- has to be **satisfied**
- **violation** = invalid solution
- penalty = ∞

Types of Constraints

Soft Constraint

- aka **preferences**
- *example*: prefer red over green
- has associated **cost** or **penalty**
- *may or may not* be satisfied
- constrained **optimization** problem

CSP vs COP

Satisfaction

- **all** constraints must be **satisfied**

Optimization

- *not all* constraints might be satisfied
- *best case*: total penalty = 0 (all satisfied)
- find solution that **minimizes** the *total penalty*

Implementing Constraints

- *Input*: assignment / partial solution
- *Output*: Pass / Fail
- Only consider values of **assigned** variables
- Can add *feasibility checking* and *pruning* functions

Solving CSP

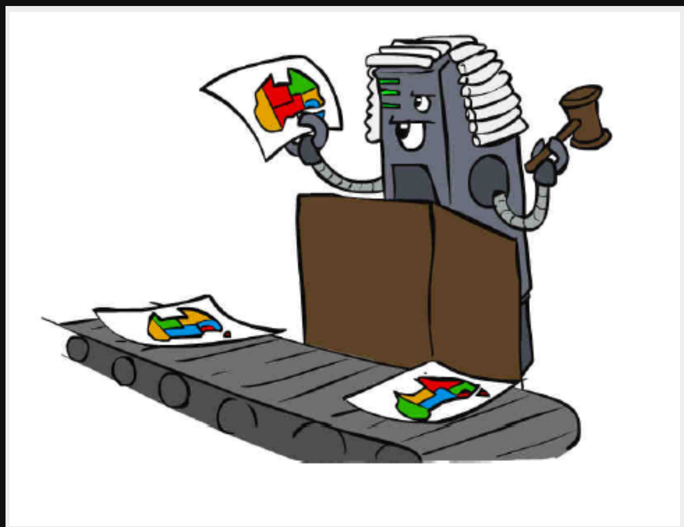
Standard Search Formulation

- Model CSP as a **Standard Search** problem
- **States** → values assigned so far (*partial assignments*)
- **Start state**: *empty assignment*

Standard Search Formulation

- **Successor fn**: assign value to unassigned variable (*extend partial assignment*)
- **Goal test**: current assignment is **complete** (no unassigned), **satisfies all** constraints

Standard Search Formulation



Backtracking Search

- Basic **uninformed algorithm** for CSP
- *Idea 1*: Extend one variable at a time
- *Idea 2*: Check constraints as you go

Backtracking: Idea 1

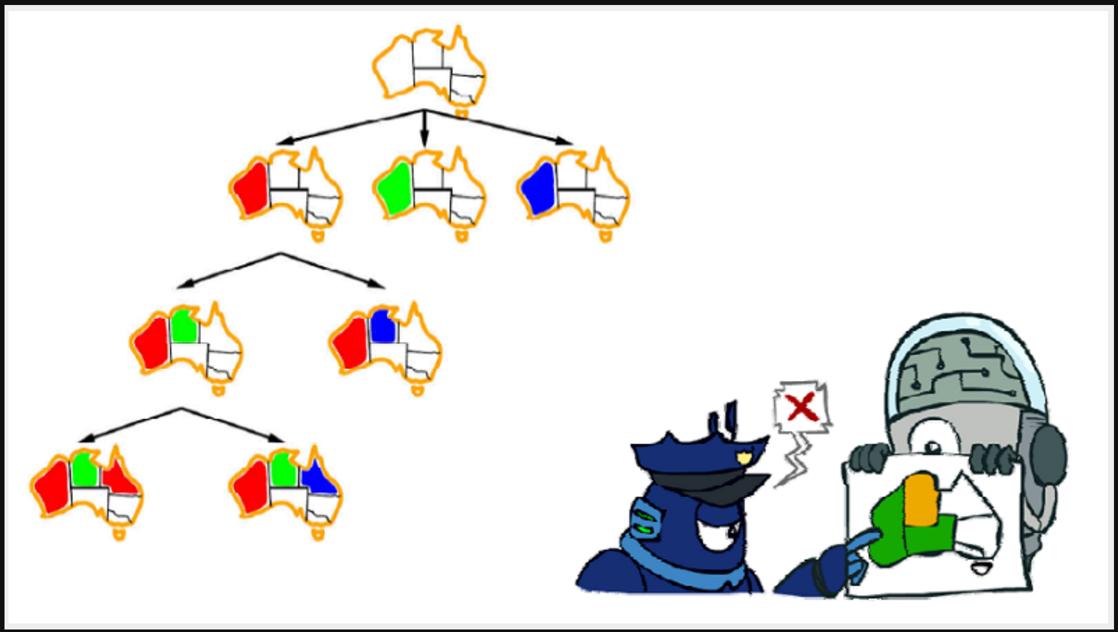
- Only need to consider assignments to **one variable** at each step
- Variable assignments are **commutative**
- e.g. [WA=**r**, NT=**g**] is same as [NT=**g**, WA=**r**]
- Fix the **variable ordering**

Backtracking: Idea 2

- **Check goal test** at *each step*
- Only consider **values** that **do not conflict** with previous assignments
- **Backtrack** as soon as we **violate** constraint

Backtracking Search

DFS + **variable ordering** + **fail-on-violation**



Demo: N-Queens, N = 4

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
```

Backtracking Search

1. *Start*: empty assignment
2. Choose **unassigned variable**
3. For each **value** in domain, try var=value
4. If **fail**, *backtrack*
5. Repeat until solution found or search tree exhausted

Backtracking Search

- CSP: **NP-Hard** in general
- *Worst-case*: **$O(b^n)$**, like DFS (*exponential*)
- **b** = branching factor = domain size
- **n** = no. of variables
- How can we improve this?

Improving Backtracking

- **Filtering**: can we remove values that will lead to inevitable failures earlier?
- **Ordering**: which variable to assign next? in what order should values be tried?
- **Avoiding thrashing**: can we avoid doing the same mistakes?

Filtering

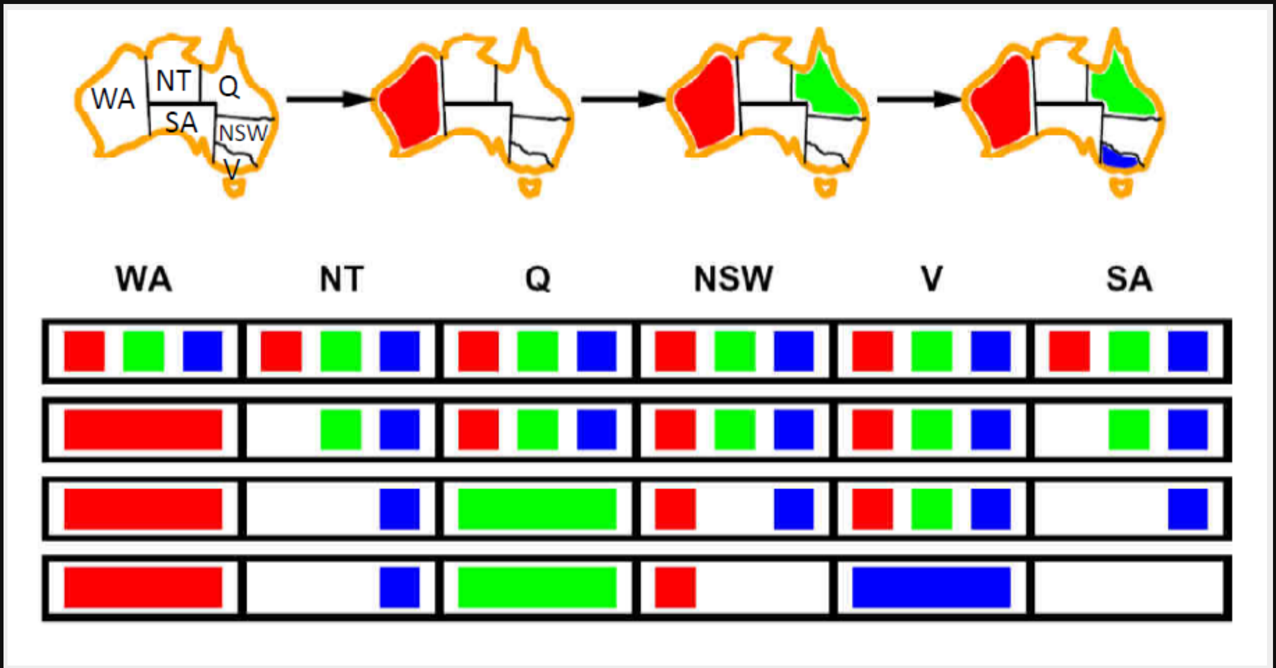
Filtering

- **Inference**: remove values that will lead to **inevitable failure**
- Keep track of *domains* of **unassigned variables** and *cross off* **bad options**
- **Early detection** of *eventual dead-ends*

Forward Checking

- **Remove values** that **violate** a constraint when added to existing assignment
- If a variable's domain becomes empty, no solution possible → **backtrack early**

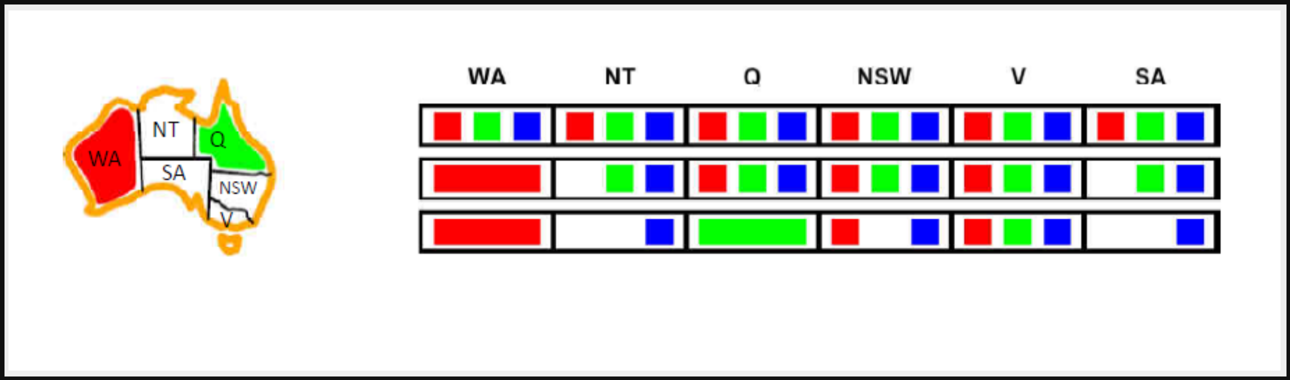
Forward Checking



Forward Checking

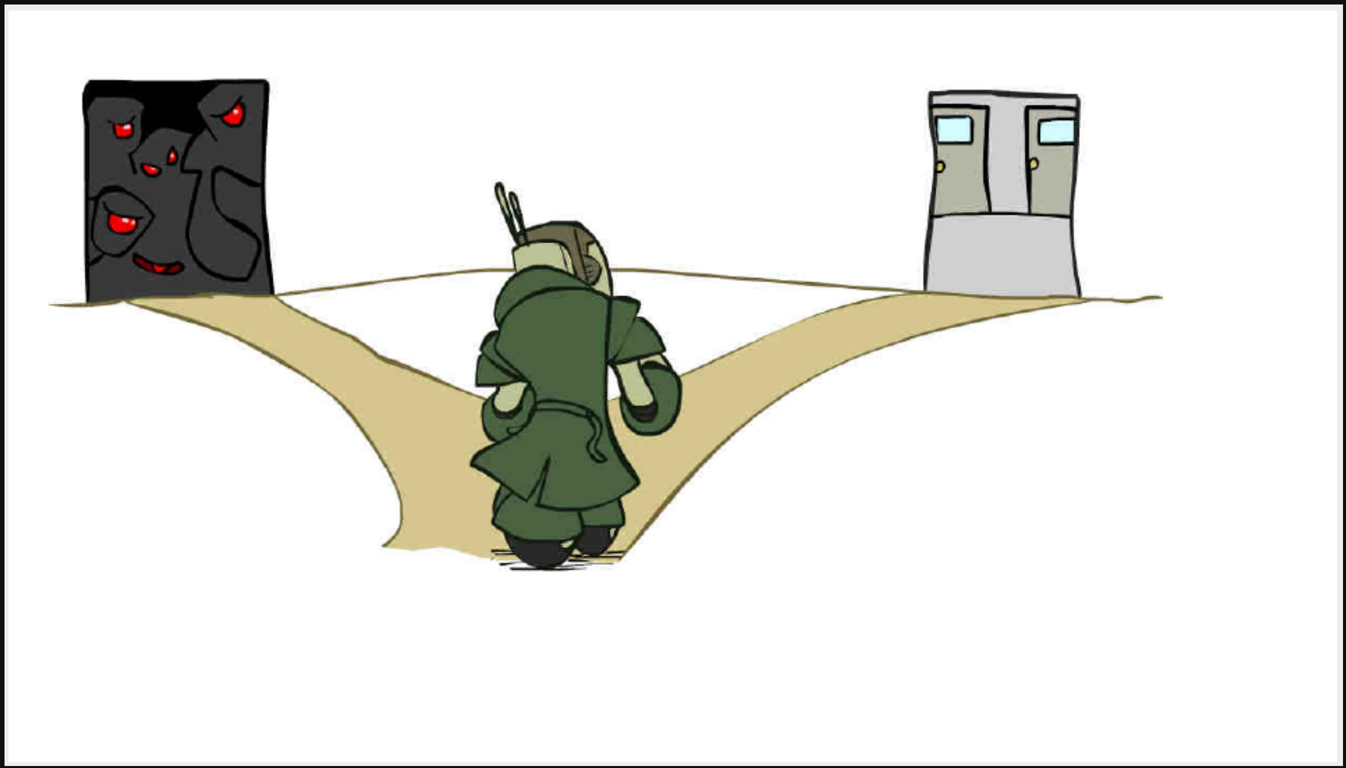
- Propagates *information* from assigned variables to unassigned variables
- Does not provide early detection for all types dead-ends / failures
- **Advanced filtering**: Arc Consistency

Limitation



NT and SA cannot both be blue!

Ordering



Ordering

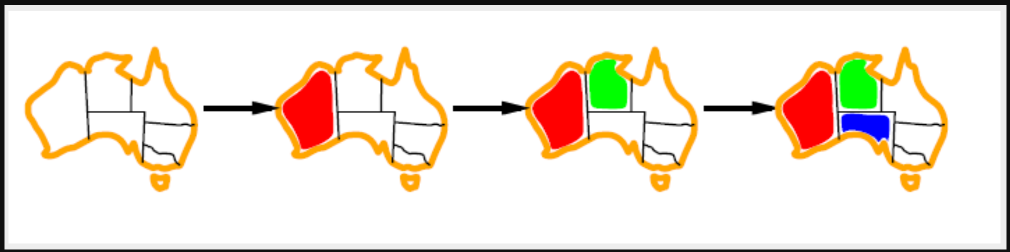
- Which **variable** to assign next?
- In what order should **values** be tried?
- Use **ordering heuristics**

Variable Ordering

Minimum Remaining Values (MRV)

- choose variable with **fewest legal values** left in domain
- fail-fast ordering

Minimum Remaining Values

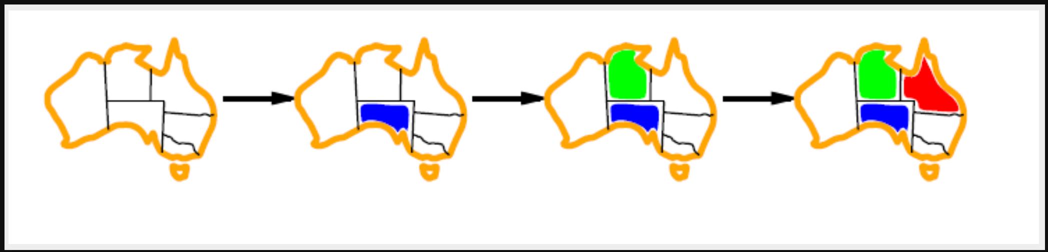


Variable Ordering

Degree Heuristic (DH)

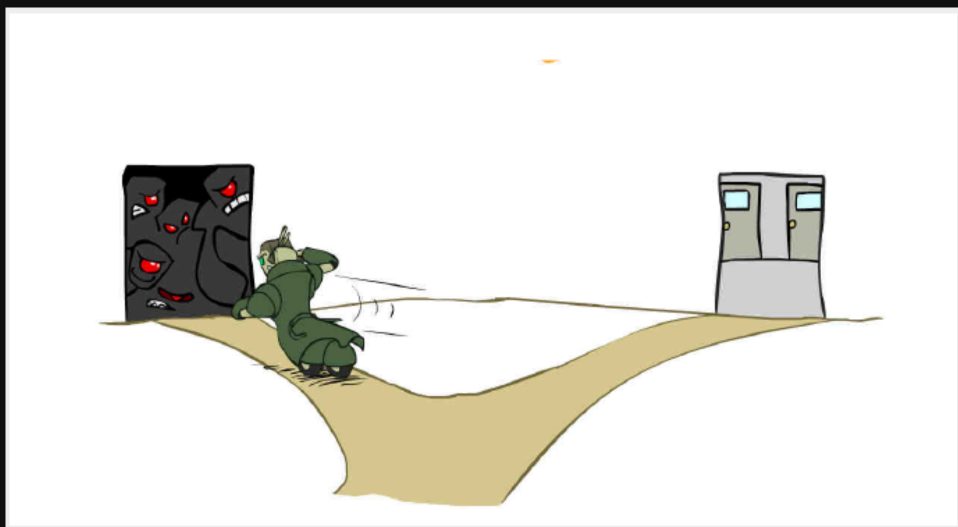
- *tie-breaker* for MRV variables
- choose variable with **most constraints** on *remaining* variables
- fail-fast ordering

Degree Heuristic



Variable Ordering

Fail-fast approach

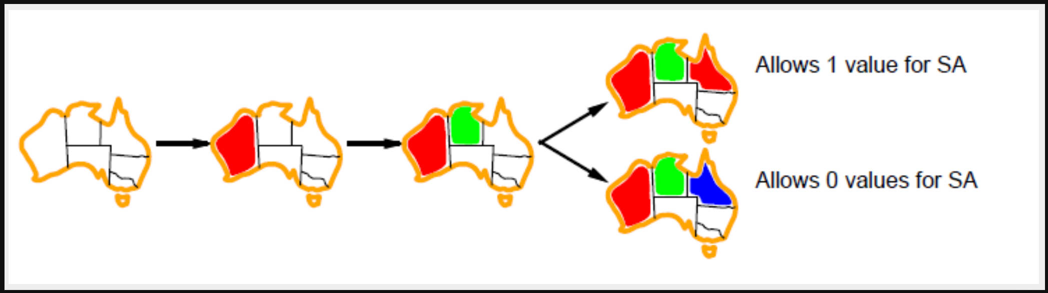


Value Ordering

Least Constraining Value (LCV)

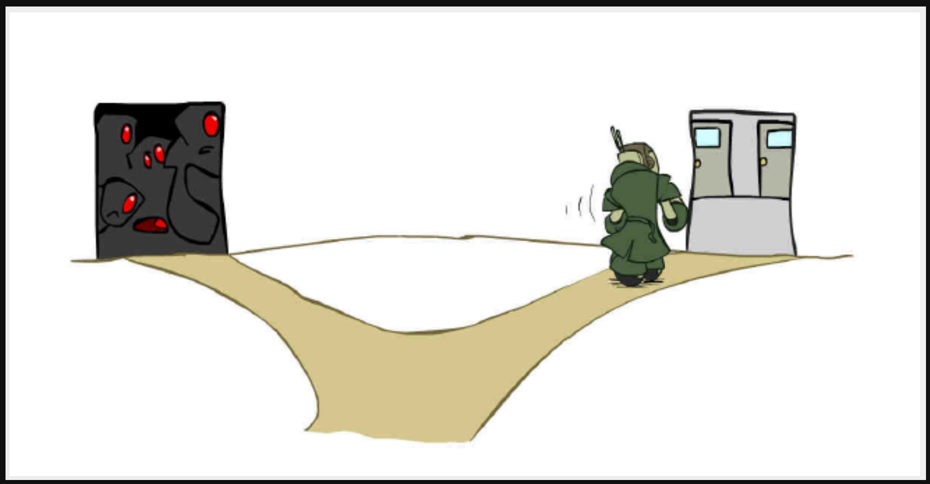
- given variable, choose value that **rules out fewest values** in remaining variables
- choose value that gives the *best chance* for solution to be extended
- **more computation** (e.g. re-run filtering)

Least Constraining Value



Value Ordering

Choose best chance for survival



Ordering

N-Queens:

- Basic *backtracking* can solve for **N=25**
- Using *ordering heuristics* makes **N=1000** queens *feasible*

Ordering

- MRV, DH, LCV apply to **CSP** + **one solution**
- If *COP* or looking for *all solutions*, use different heuristics

Thrashing

- **Repeated failure** due to the *same reason*
- Basic backtracking doesn't identify *root* of problem: **conflicting variables**
- Search in *different parts* of the search space keep failing for the *same reason*

Thrashing Solution

Intelligent Backtracking

- aka *non-chronological backtracking*
- go back **directly** to variable that caused failure, instead of *previous variable*
- *example*: backjumping

Redundant Work

- Even if **conflicting variables** are identified, **not remembered** (no memory)
- *Same conflict* in subsequent computation will still occur
- *Solution*: **NoGoods recording**

NoGoods Recording

- Keep track of **combinations of values** that will always lead to **failure**
- Make the mistake once & **record** it

NoGoods Recording

- **Avoid** exploring **same mistake** later in other parts of search tree
- *Issues*: **updating** and **querying** nogoods database

Summary

- **Constraint Satisfaction Problems**
  - **Variables**
  - **Domains**
  - **Constraints**
- Constraint **Satisfaction** vs **Optimization**
- **Backtracking** algorithm

Summary

Improving Backtracking:

- **Filtering**: forward checking
- **Ordering**: variable (**MRV**, **DH**), value (**LCV**)
- **Avoiding thrashing**: backjumping, nogoods recording

Next Meeting

- Local Search
- Hill Climbing
- Simulated Annealing
- Tabu Search

Assignment 3

- By pair
- 1 CSP, 1 COP
- Description, Variables, Domains, Constraints
- Post answers on Facebook Group thread
- No duplication, FCFS

Announcements

Course Requirements Update:

- 15% - MP 1 (**deadline: Monday**)
- **15%** - MP 2
- 20% - MP 3
- 15% - MP 4
- **15%** - Assignments / Quizzes
- 20% - Final Project

References

- *Artificial Intelligence: A Modern Approach, 3rd Edition*, S. Russell and P. Norvig, 2010
- CS 188 Lec 4,5 slides, Dan Klein, UC Berkeley

Questions?