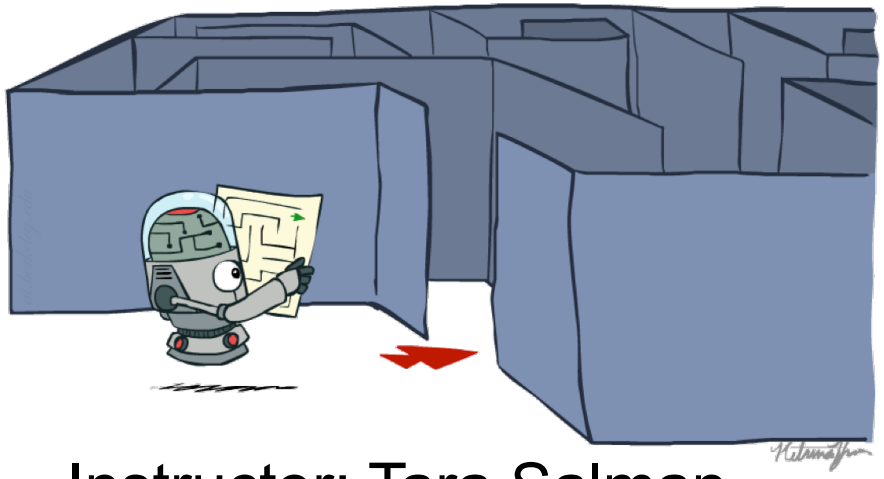
Announcements

- Homework 0 and Project 0 were due today
  - Project 0 is mandatory (part of project grades)
  - Homework 0 is optional (2 points by max are added to the homework grades. That is homework grade can be 12/10)
- Homework 1 and Project 1 will be out today
  - Due on Tuesday September 21
  - Homework 1 is already on Gradescope
  - Project 1 will be out today on Blackboard
- Review Questions and Piazza
  - Review Questions are at the end of lecture slides. Make sure you do them
  - Post questions about them on Piazza. We will be monitoring them. Your classmates will answer

CS 3568: Intelligent Systems

Search (Part 4)



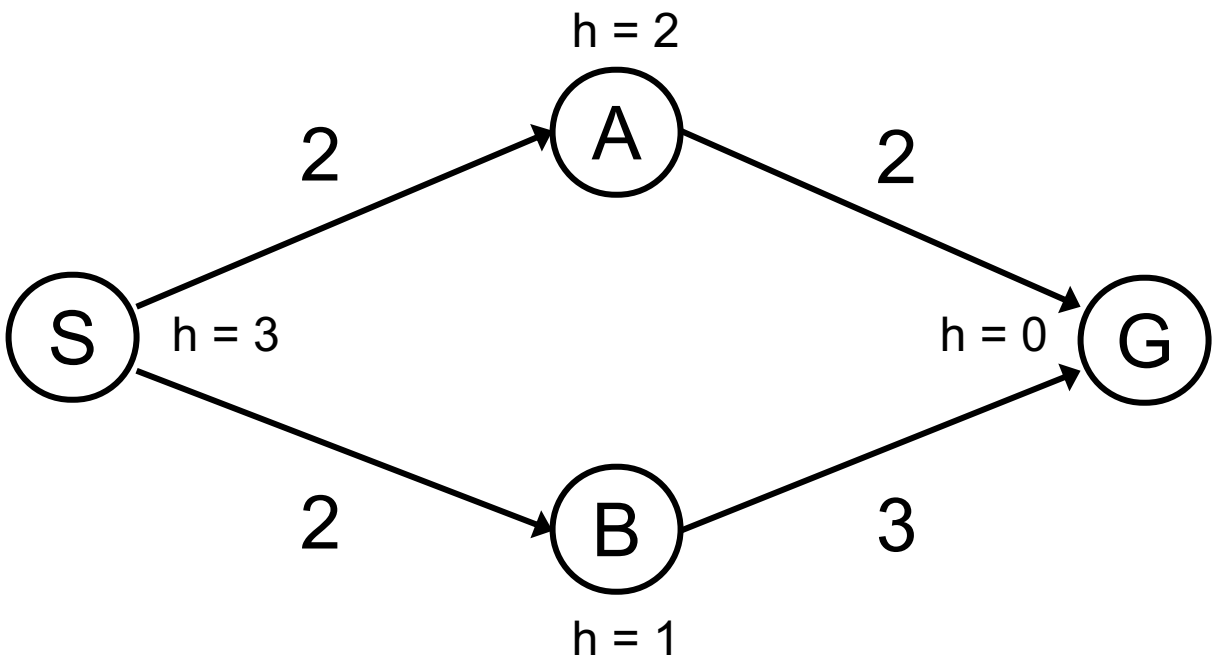
Instructor: Tara Salman

Texas Tech University

Computer Science Department

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

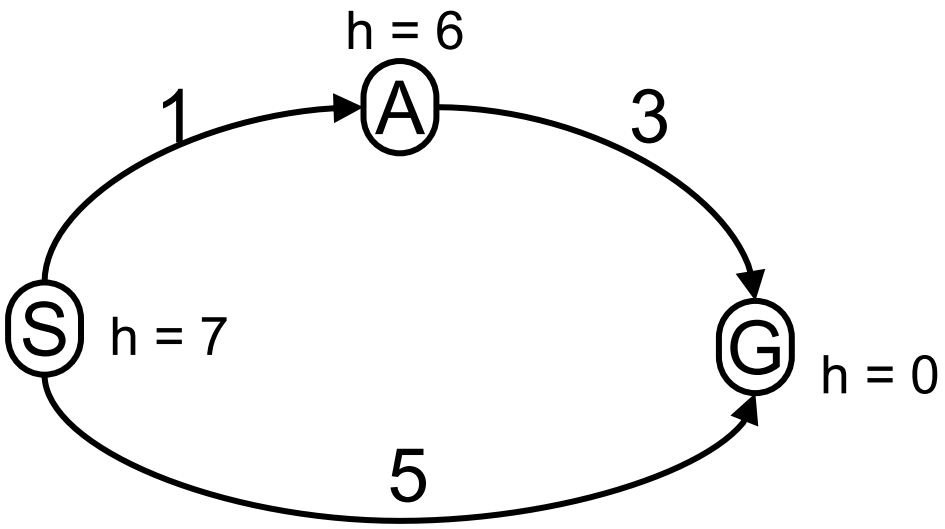
When should A* terminate?



f(n)=g(n)+h(n)
g(n) is cumulative cost. E.g.,
g(G)=2+2=4 (through A)
h(n) is non-cumulative. E.g, h(G)=0
f(n) is non-cumulative. f(G)=4+0=0 (through A)

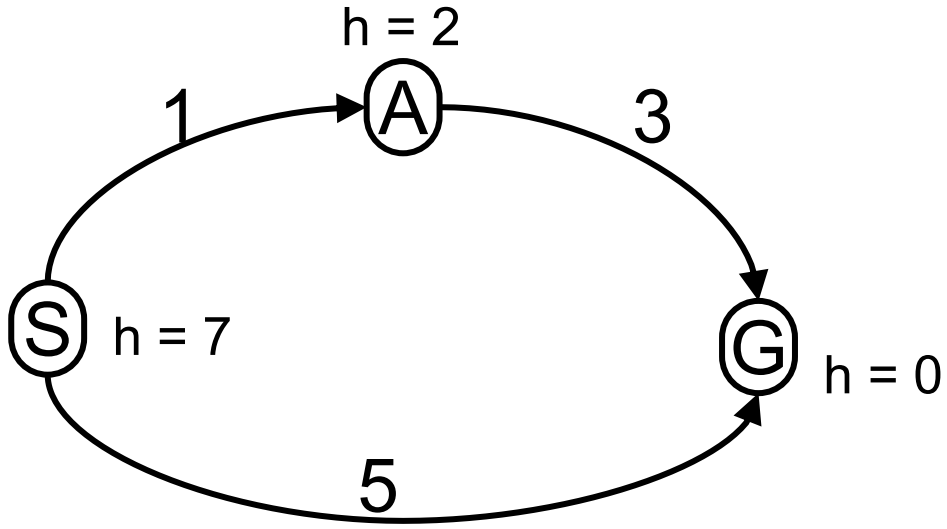
➔ **f(G)≠ f(A)+ g(G)+h(0)**

Is A* Optimal?



Non-admissible

F(A)=7
F(G) non-stop=5
F(G) final=5



admissible

F(A)= 3
F(G) non-stop=5
F(G) final=4

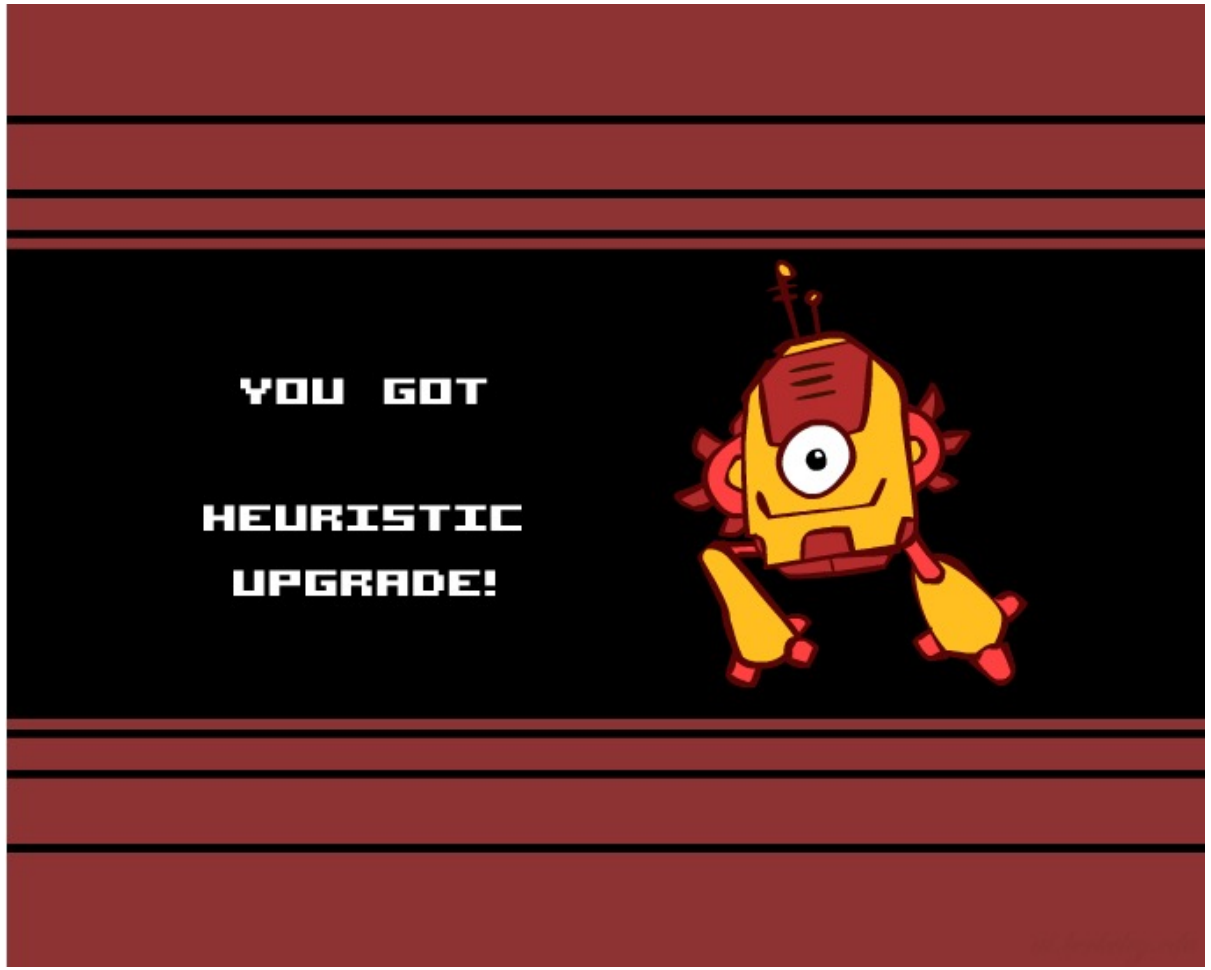
Last Lecture

- A*
  - Optimality
  - Examples
  - Comparisons

Today's Lecture

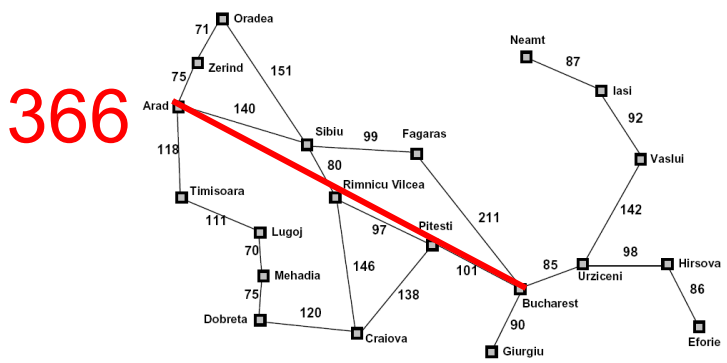
- How to choose heuristics
- Graph Search

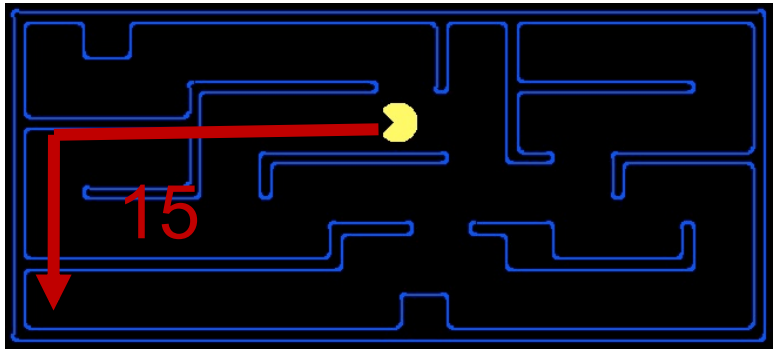
Creating Heuristics



Creating Admissible Heuristics

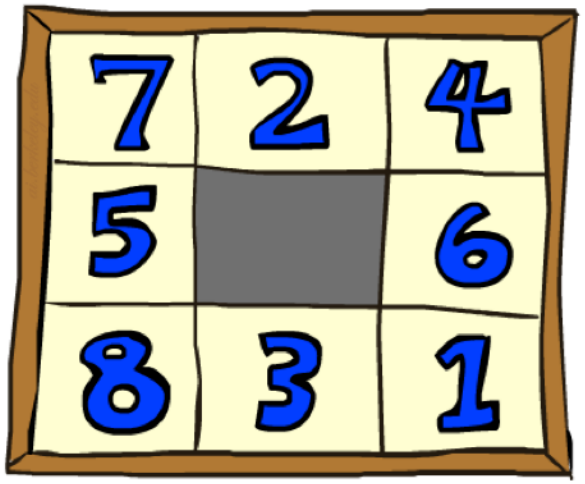
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



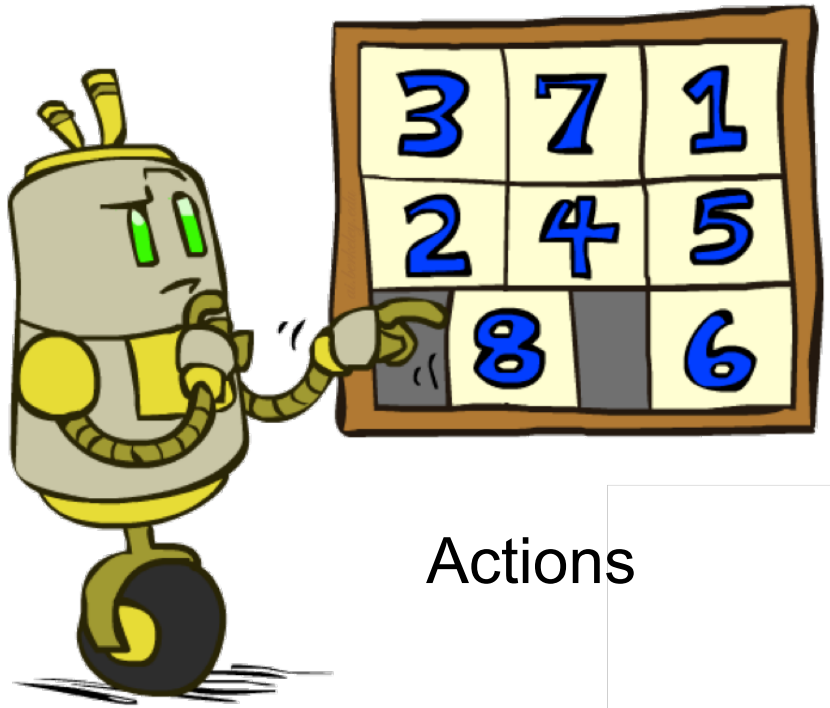


- Inadmissible heuristics are often useful too

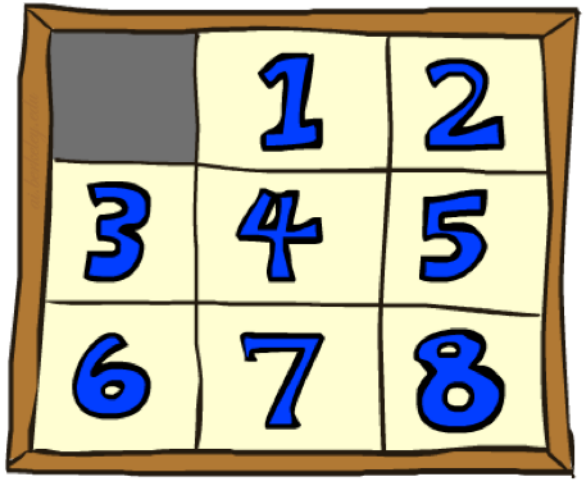
Example: 8 Puzzle



Start State



Actions

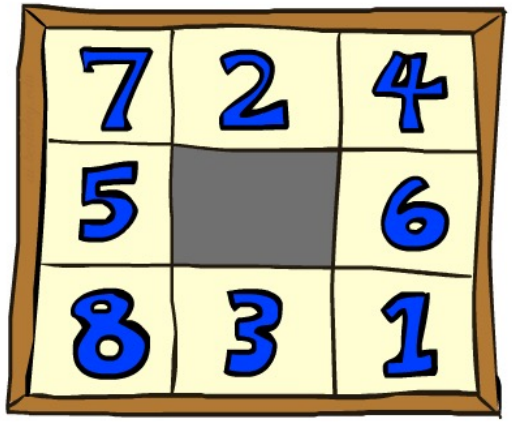


Goal State

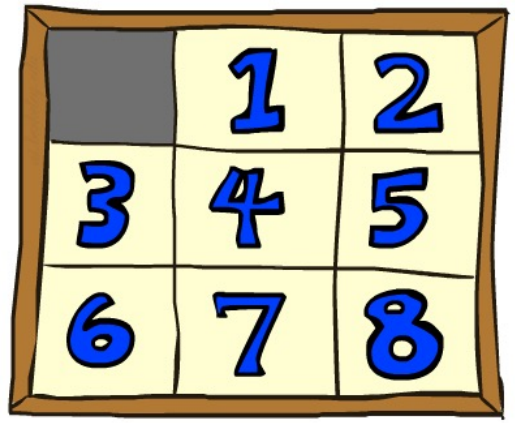
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

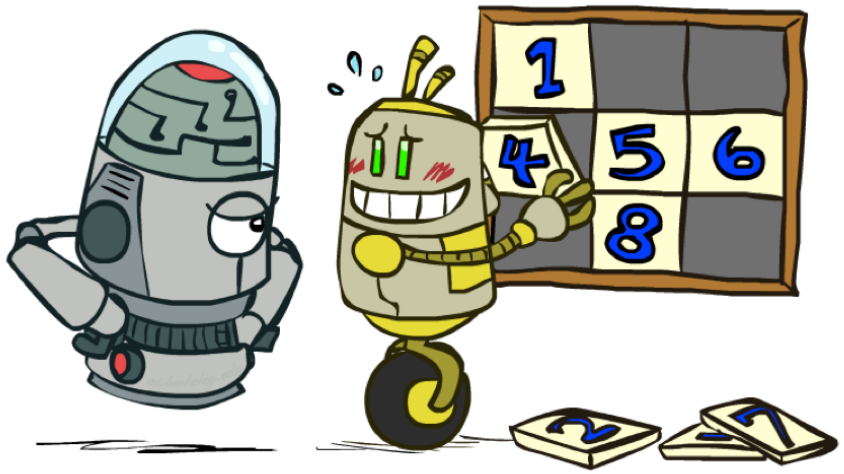
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



Start State



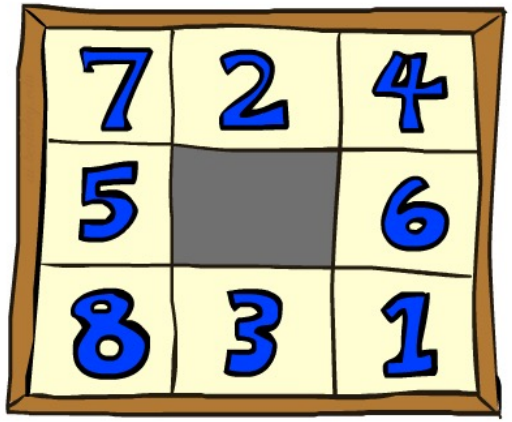
Goal State



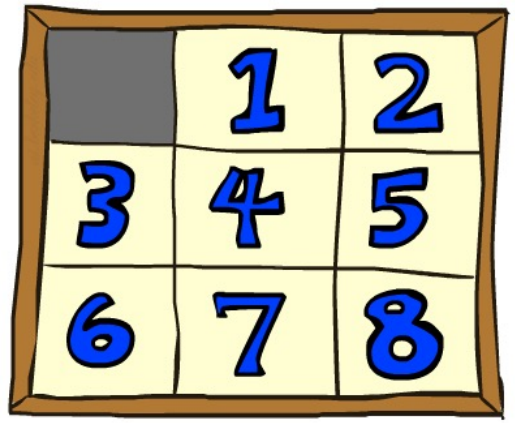
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- h(start) = 3 + 1 + 2 + … = 18



Start State

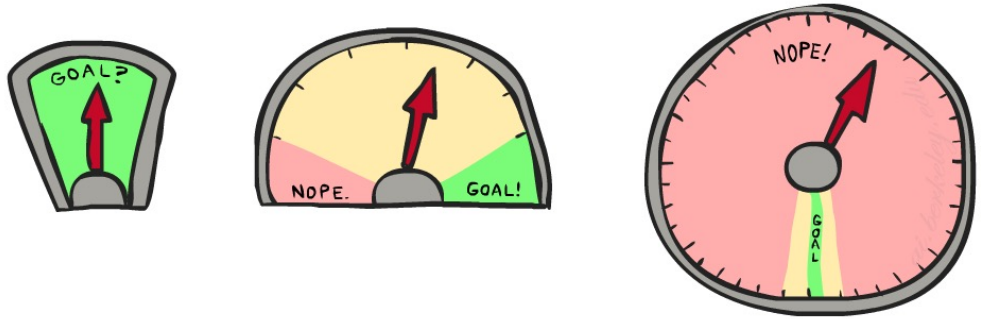


Goal State

	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Semi-Lattice of Heuristics

Trivial Heuristics, Dominance

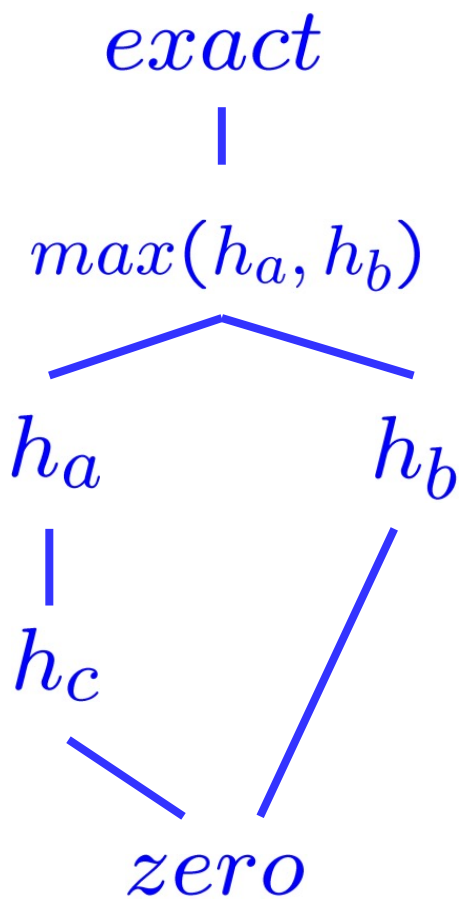
- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

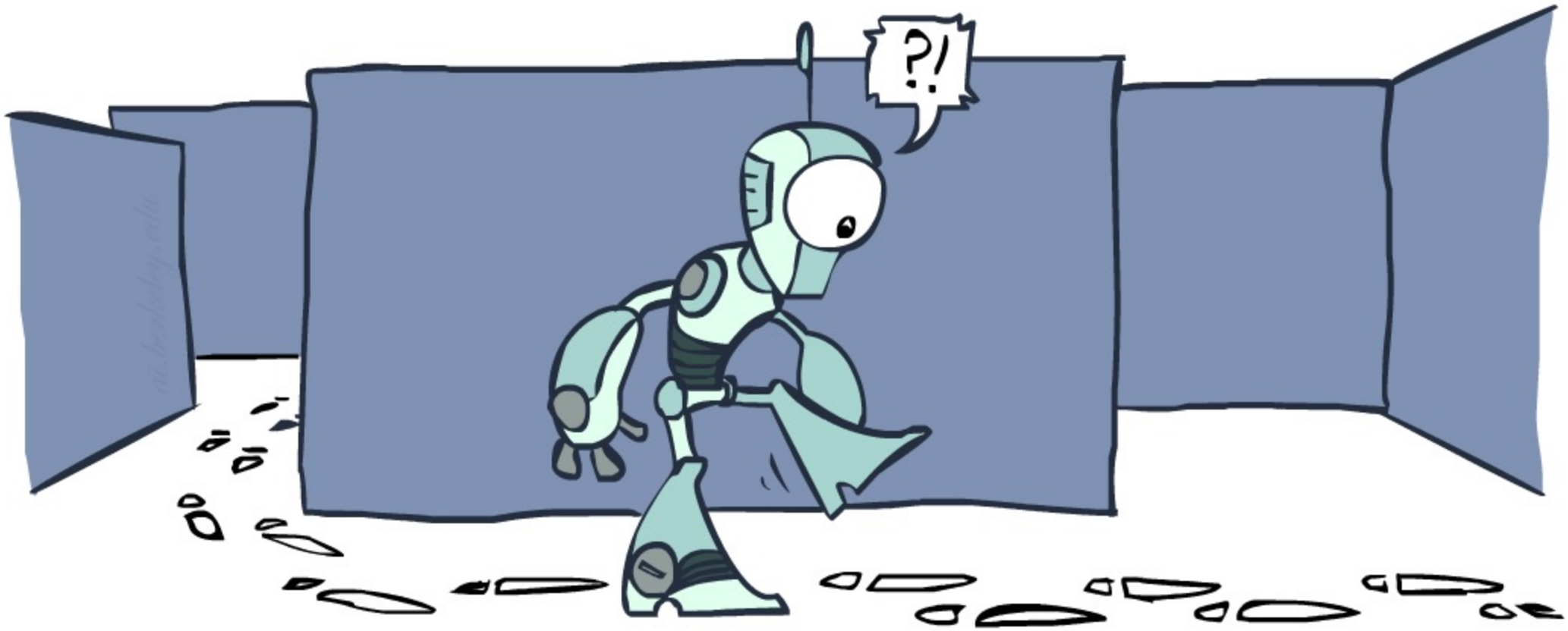
- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

$$h(n) = max(h_a(n), h_b(n))$$

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic

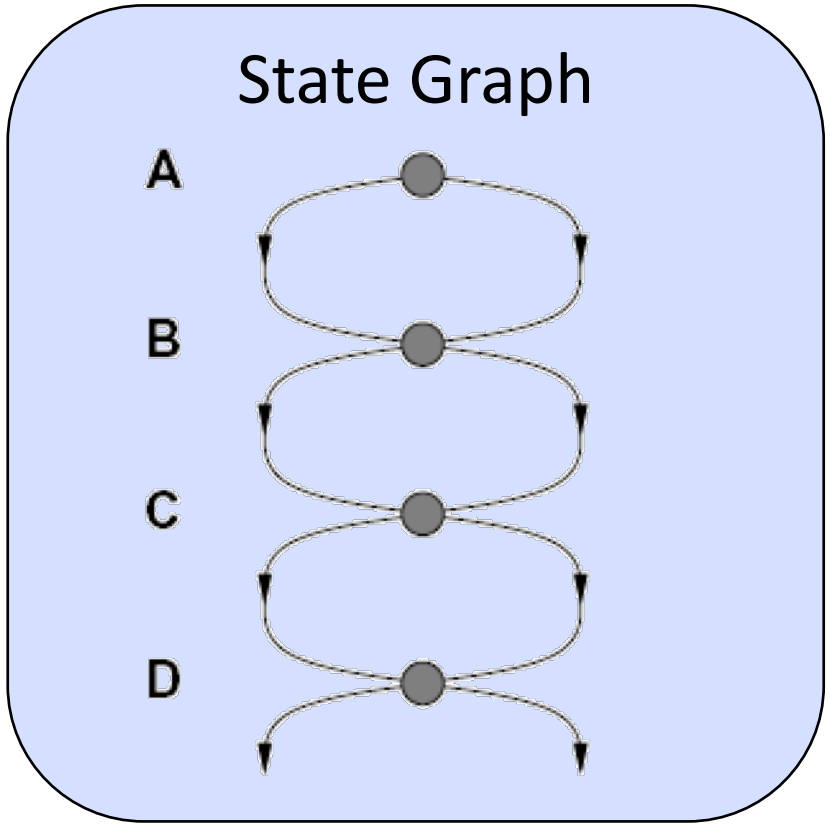


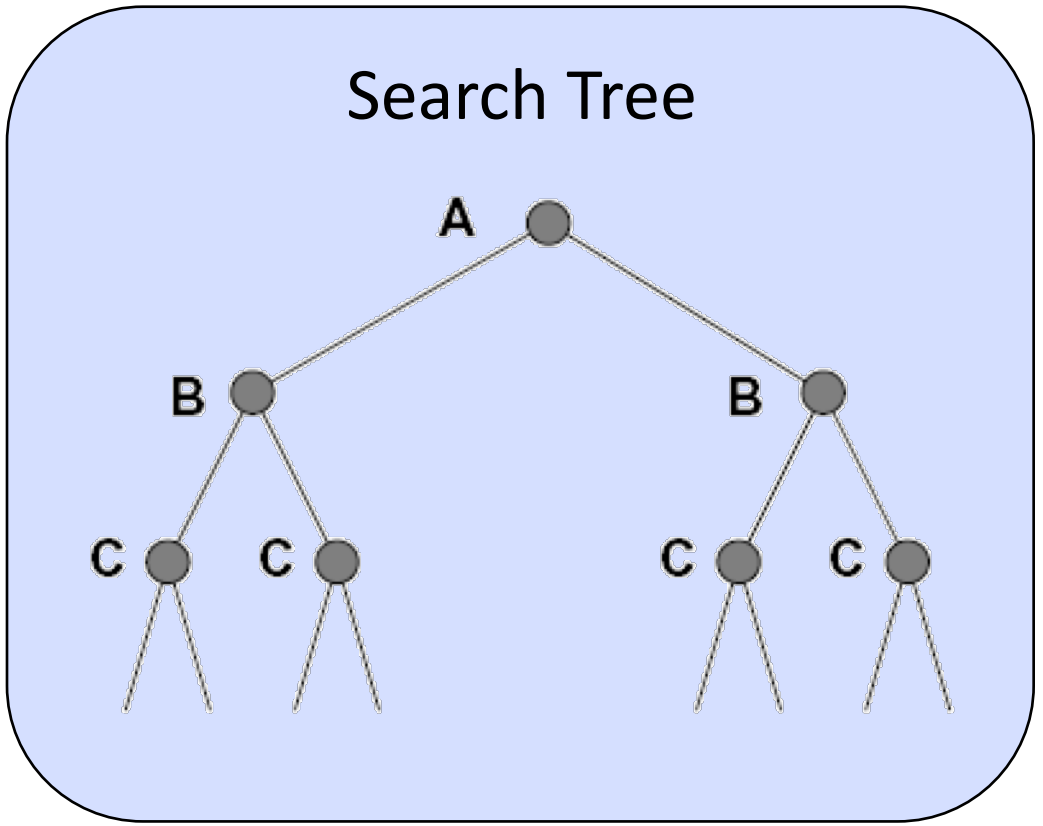
Graph Search



Tree Search: Extra Work!

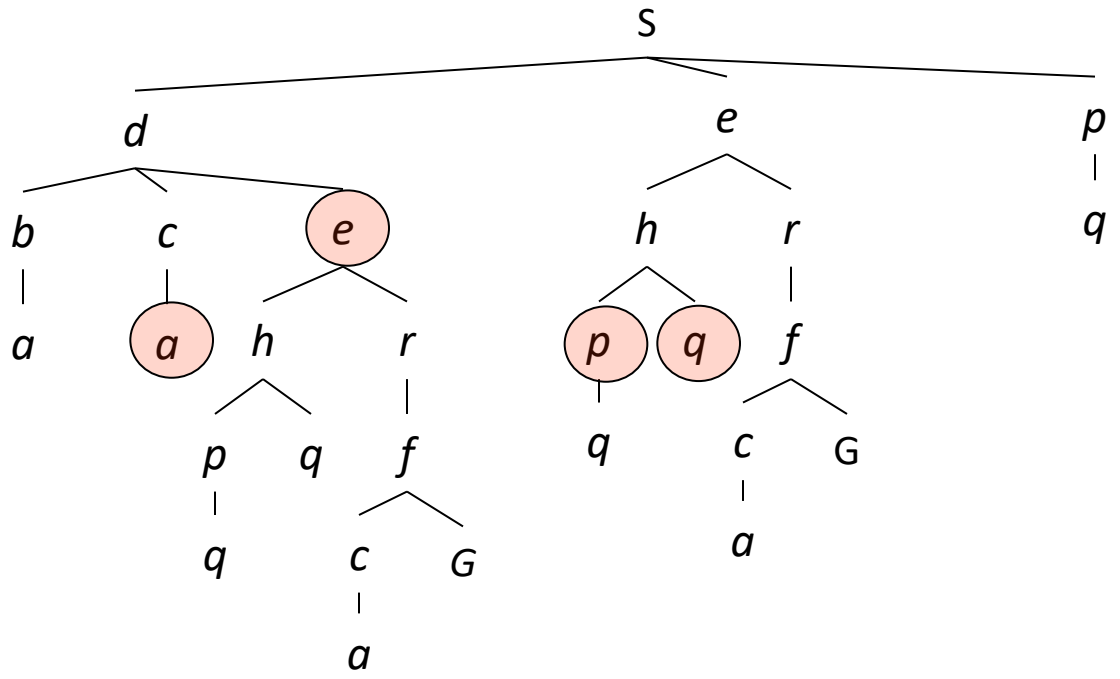
- Failure to detect repeated states can cause exponentially more work.





Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

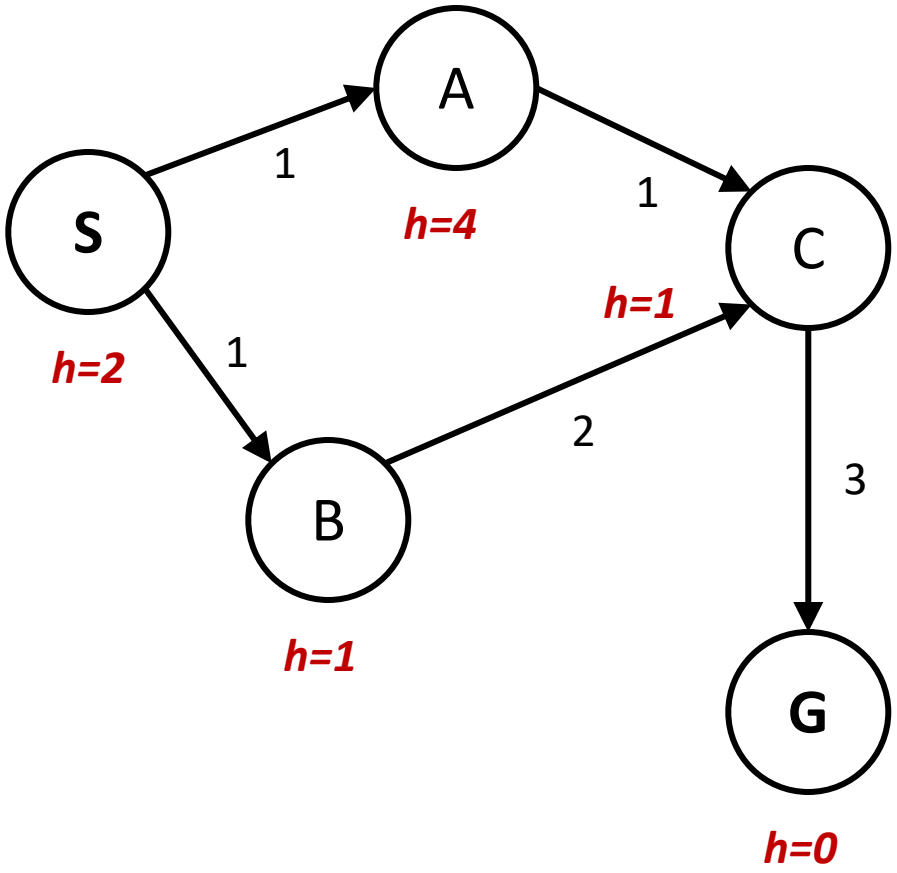


Graph Search

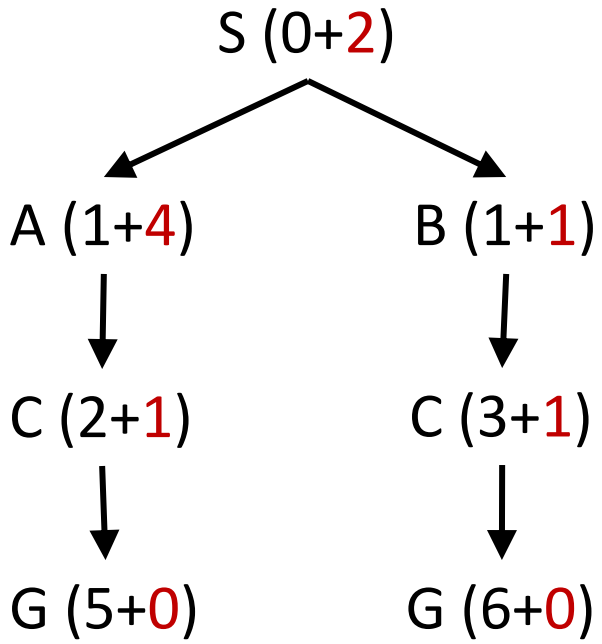
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?

A* Graph Search Gone Wrong?

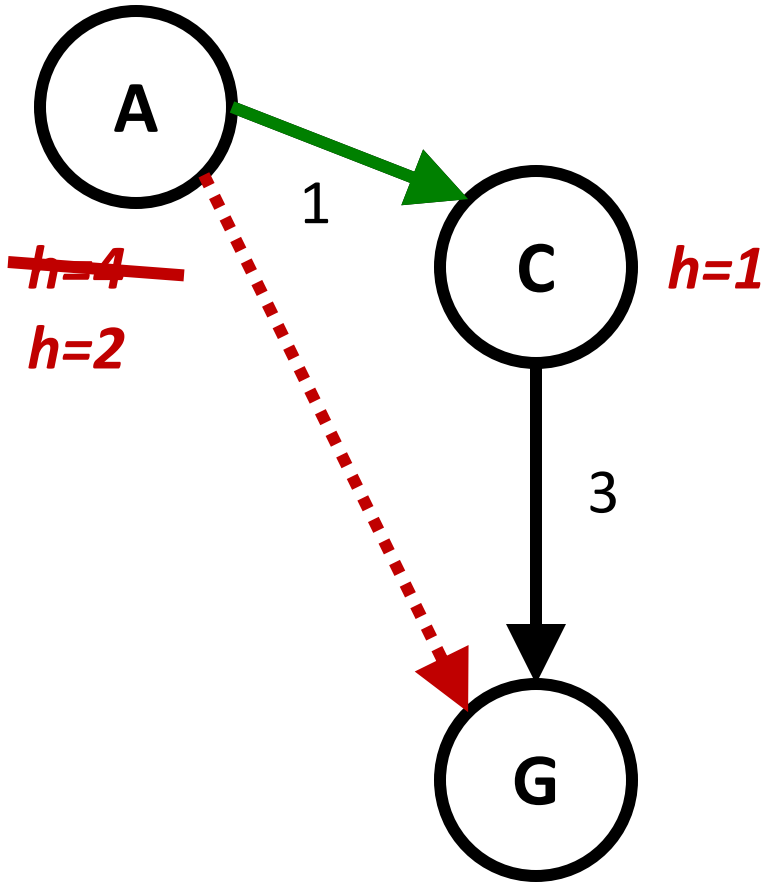
State space graph



Search tree



Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

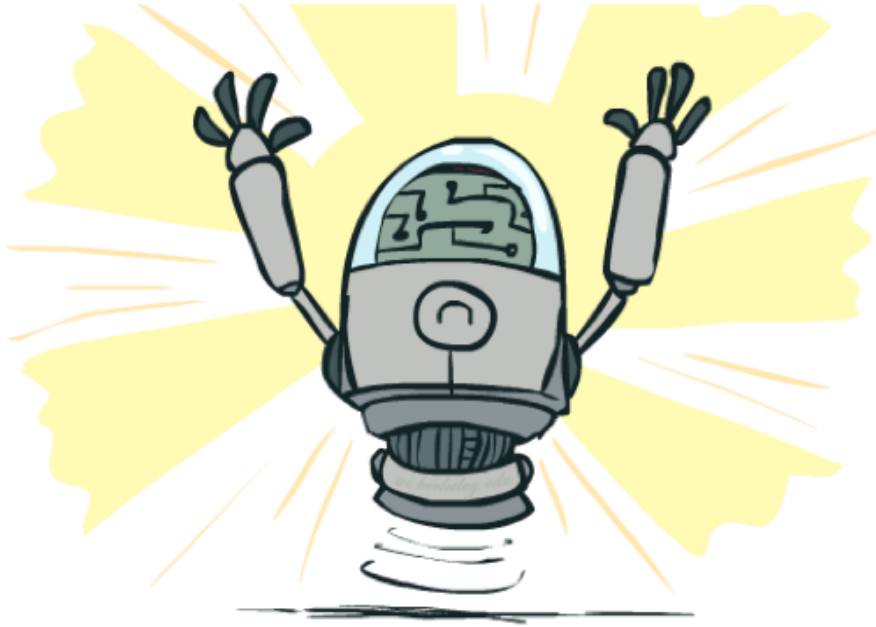
    h(A) ≤ actual cost from A to G
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)
- Consequences of consistency:
  - The f value along a path never decreases

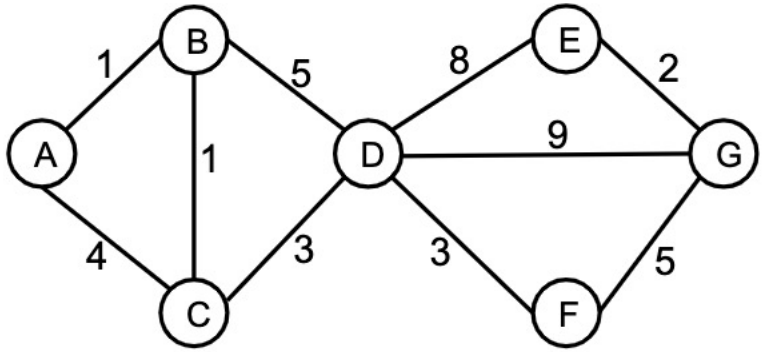
    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



Review Question

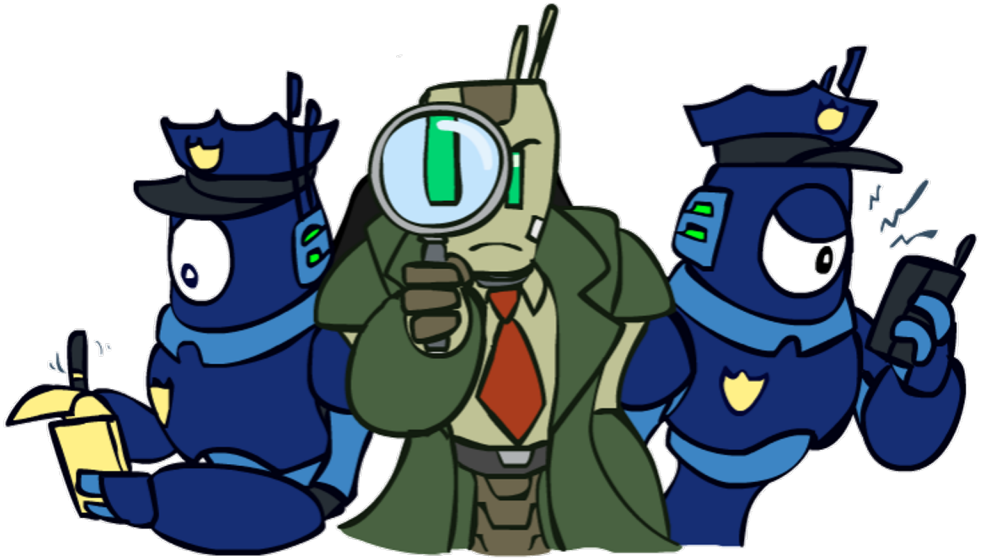


Node	A	B	C	D	E	F	G
h_3	10	?	9	7	1.5	4.5	0

1. What values of $h_3(B)$ make h_3 admissible?
2. What values of $h_3(B)$ make h_3 consistent?
3. What values of h3(B) will cause A* graph search to expand node A, then node C, then node B, then node D in order?

CS 3568: Intelligent Systems

Constraint Satisfaction Problems (Part 1)



Instructor: Tara Salman

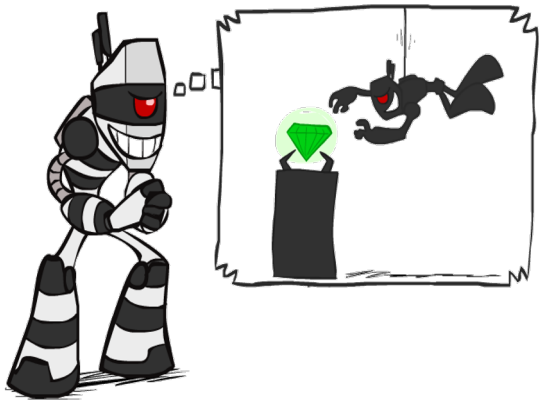
Texas Tech University

Computer Science Department

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

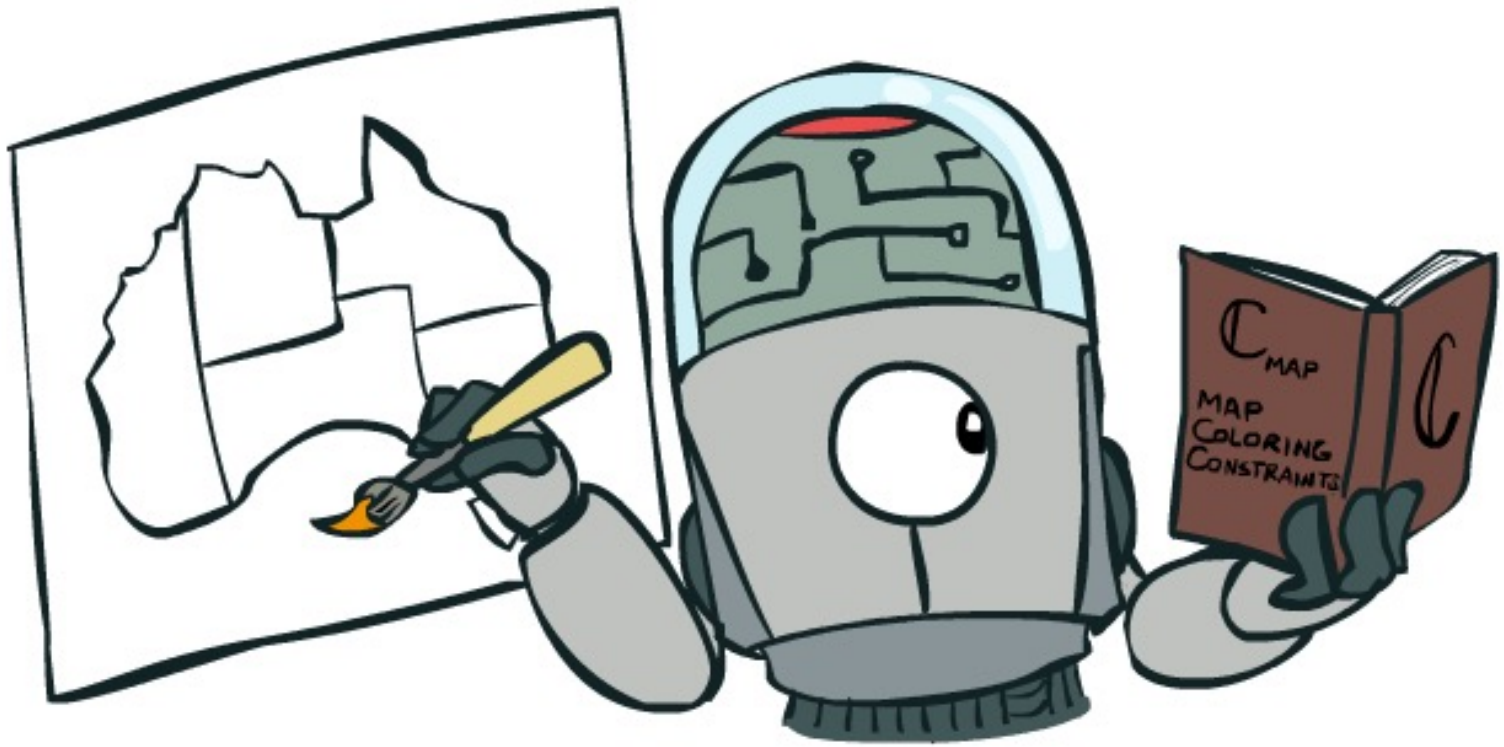
What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space
- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance



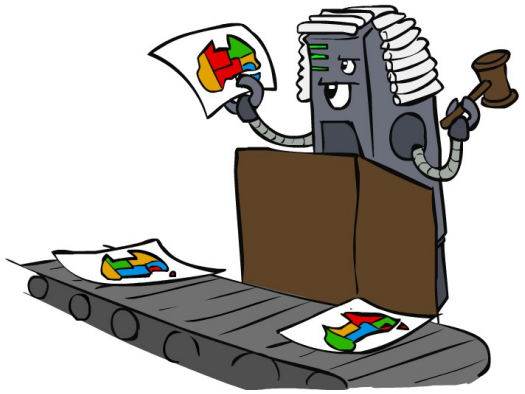
- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are a specialized class of identification problems

Constraint Satisfaction Problems

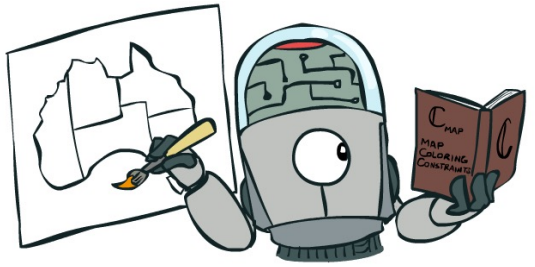


Constraint Satisfaction Problems

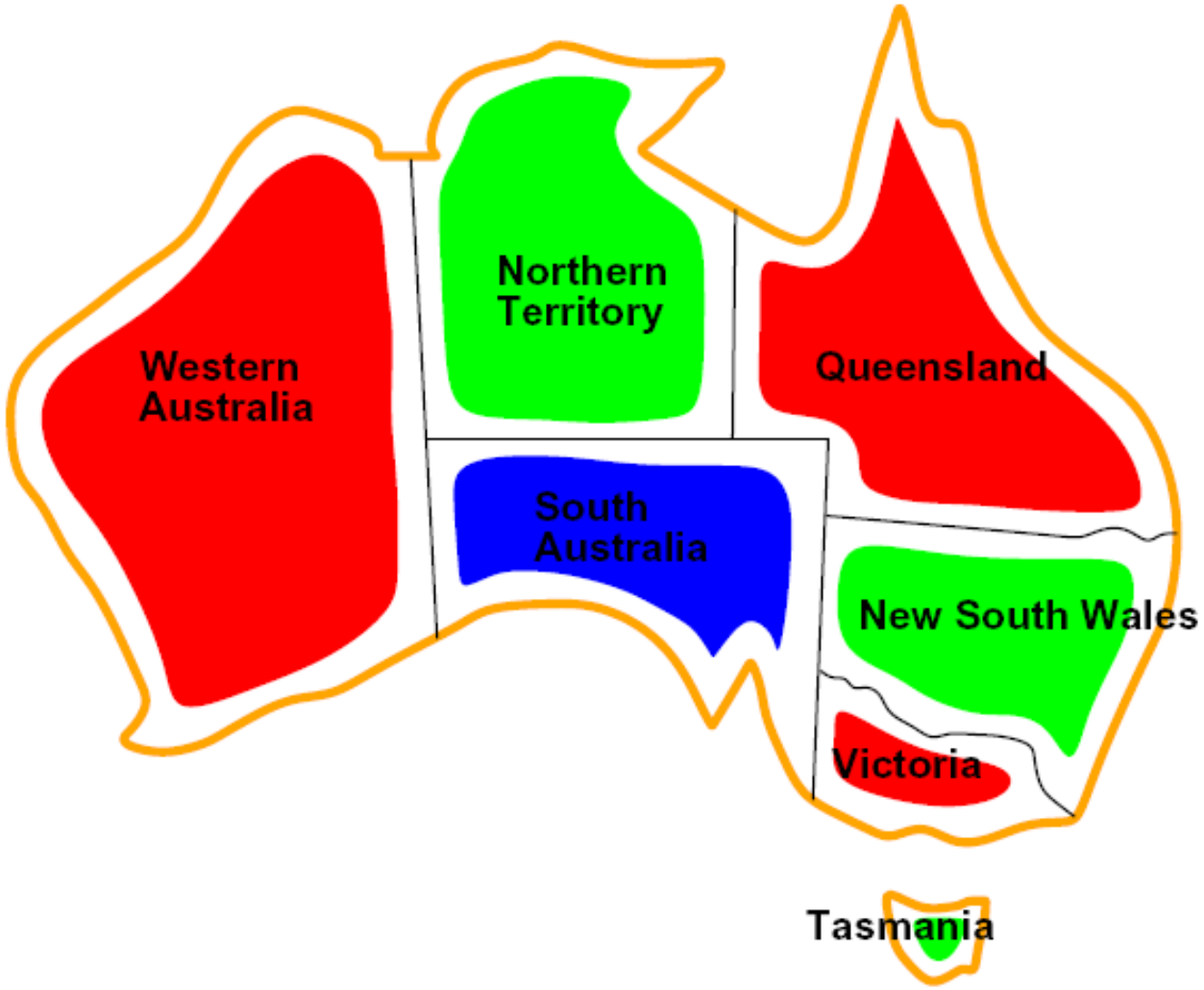
- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything



- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a 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

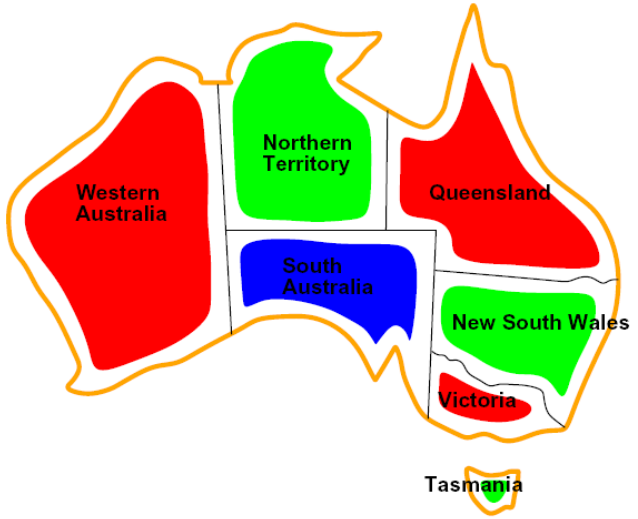
- Variables: $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$
- Domains: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors

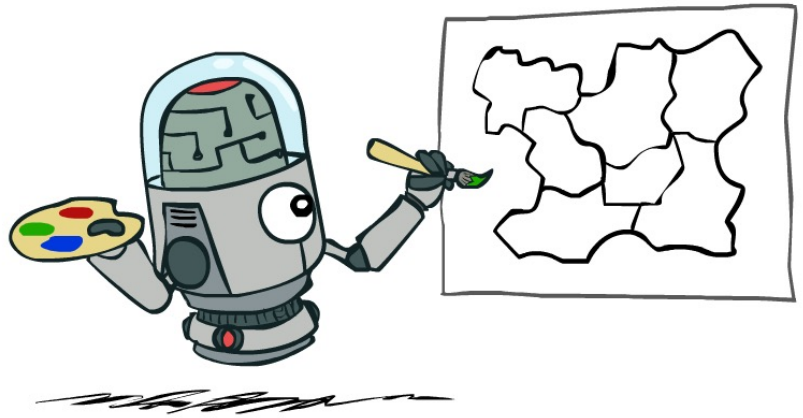
  Implicit: $WA \neq NT$

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

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

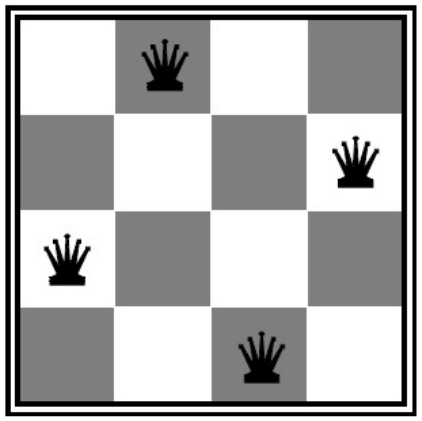
  $\{WA{=}red,\ NT{=}green,\ Q{=}red,\ NSW{=}green,$
  $V{=}red,\ SA{=}blue,\ T{=}green\}$

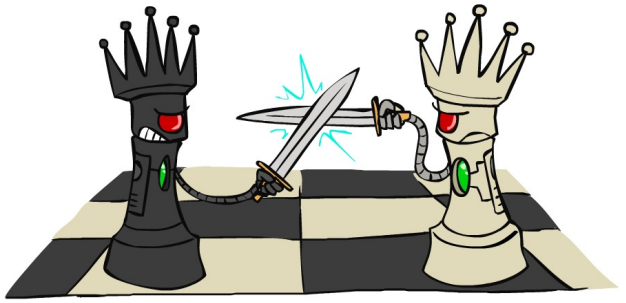




Example: N-Queens

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





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

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

Example: N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:

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

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

    $\ldots$